Unspecified and Undefined

by Olve Maudal

Strange things can, and will, happen if you break the rules of the language. In C there is a very detailed contract between the programmer and the compiler that you need to understand well. It is sometimes said that upon encountering a contract violation in C, it is legal for the compiler to make nasal demons fly out of your nose. In practice, however, compilers usually do not try to pull pranks on you, but even when trying to do their best they might give you big surprises.

In this talk we will study actual machine code generated by snippets of both legal and illegal C code. We will use these examples while studying and discussing parts of the ISO/IEC 9899 standard (the C standard).

A 90 minute session at ACCU 2013 in Bristol UK

Thursday, April 11, 2013

Warning: this presentation contains a lot of crap code. Don't try this at home.

Warning: this presentation contains a lot of crap code. Don’t try this at ~~home~~.

Warning: this presentation contains a lot of crap code. Don't try this at work.

Exercise

This code is **undefined behavior**, but what do you think actually happens if you compile, link and run it in your development environment?

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

Exercise

This code is **undefined behavior**, but what do you think actually happens if you compile, link and run it in your development environment?

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

Exercise

This code is **undefined behavior**, but what do you think actually happens if you compile, link and run it in your development environment?

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ gcc foo.c && ./a.out
```

Exercise

This code is **undefined behavior**, but what do you think actually happens if you compile, link and run it in your development environment?

foo.c

```c
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ gcc foo.c && ./a.out
12
```

Exercise

This code is **undefined behavior**, but what do you think actually happens if you compile, link and run it in your development environment?

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ gcc foo.c && ./a.out
12
$ clang foo.c && ./a.out
```

Exercise

This code is **undefined behavior**, but what do you think actually happens if you compile, link and run it in your development environment?

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ gcc foo.c && ./a.out
12
$ clang foo.c && ./a.out
11
```

Exercise

This code is **undefined behavior**, but what do you think actually happens if you compile, link and run it in your development environment?

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ gcc foo.c && ./a.out
12
$ clang foo.c && ./a.out
11
$ icc foo.c && ./a.out
```

Exercise

This code is **undefined behavior**, but what do you think actually happens if you compile, link and run it in your development environment?

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ gcc foo.c && ./a.out
12
$ clang foo.c && ./a.out
11
$ icc foo.c && ./a.out
13
```

foo.c

```c
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

It is always a good idea to add some flags for better diagnostics.

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

It is always a good idea to add some flags for better diagnostics.

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

It is always a good idea to add some flags for better diagnostics.

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ gcc -std=c99 -O -Wall -Wextra -pedantic foo.c && ./a.out
```

It is always a good idea to add some flags for better diagnostics.

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ gcc -std=c99 -O -Wall -Wextra -pedantic foo.c && ./a.out
12
```

It is always a good idea to add some flags for better diagnostics.

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ gcc -std=c99 -O -Wall -Wextra -pedantic foo.c && ./a.out
12
$ clang -std=c99 -O -Wall -Wextra -pedantic foo.c && ./a.out
```

It is always a good idea to add some flags for better diagnostics.

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ gcc -std=c99 -O -Wall -Wextra -pedantic foo.c && ./a.out
12
$ clang -std=c99 -O -Wall -Wextra -pedantic foo.c && ./a.out
11
```

It is always a good idea to add some flags for better diagnostics.

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ gcc -std=c99 -O -Wall -Wextra -pedantic foo.c && ./a.out
12
$ clang -std=c99 -O -Wall -Wextra -pedantic foo.c && ./a.out
11
$ icc -std=c99 -O -Wall -Wextra -pedantic foo.c && ./a.out
```

It is always a good idea to add some flags for better diagnostics.

foo.c

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ gcc -std=c99 -O -Wall -Wextra -pedantic foo.c && ./a.out
12
$ clang -std=c99 -O -Wall -Wextra -pedantic foo.c && ./a.out
11
$ icc -std=c99 -O -Wall -Wextra -pedantic foo.c && ./a.out
13
```

```c
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

Can we learn anything about C by studying code like this?

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

Can we learn anything about C by studying code like this?

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

YES!

Can we learn anything about C by studying code like this?

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

YES!

If your answer is a definitive NO, then I am afraid that the rest of this presentation is probably a waste of your time.

```c
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

```c
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

First let us agree on the theoretical part first.

```c
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

First let us agree on the theoretical part first.

The code is undefined behavior because it tries to modify a value twice between two sequence points. The rules of sequencing has been violated. The standard says:

```c
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

First let us agree on the theoretical part first.

The code is undefined behavior because it tries to modify a value twice between two sequence points. The rules of sequencing has been violated. The standard says:

[6.5 Expressions]

...

Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression. Furthermore, the prior value shall be read only to determine the value to be stored.

The grouping of operators and operands is indicated by the syntax. Except as specified later (for the function-call (), &&, ||, ?:, and comma operators), the order of evaluation of subexpressions and the order in which side effects take place are both unspecified.

...

```c
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

First let us agree on the theoretical part first.

The code is undefined behavior because it tries to modify a value twice between two sequence points. The rules of sequencing has been violated. The standard says:

[6.5 Expressions]

...

Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression. Furthermore, the prior value shall be read only to determine the value to be stored.

The grouping of operators and operands is indicated by the syntax. Except as specified later (for the function-call (), &&, ||, ?:, and comma operators), the order of evaluation of subexpressions and the order in which side effects take place are both unspecified.

...

It takes a while to really understand those words. However, the key thing is the last sentence, which is basically saying: **the order of evaluation is mostly unspecified**. The rest is just a necessary consequence of this particular "language feature".

```c
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

First let us agree on the theoretical part first.

The code is undefined behavior because it tries to modify a value twice between two sequence points. The rules of sequencing has been violated. The standard says:

[6.5 Expressions]

...

Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression. Furthermore, the prior value shall be read only to determine the value to be stored.

The grouping of operators and operands is indicated by the syntax. Except as specified later (for the function-call (), &&, ||, ?:, and comma operators), the order of evaluation of subexpressions and the order in which side effects take place are both unspecified.

...

It takes a while to really understand those words. However, the key thing is the last sentence, which is basically saying: **the order of evaluation is mostly unspecified**. The rest is just a necessary consequence of this particular "language feature".

Since evaluation order here is unspecified, the expression does not make sense, and therefore the standard just says that this is **undefined behavior**.

```c
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

ASIDE

Foo.java

```java
public class Foo {
    public static void main(String args[]) {
        int v[] = {0,2,4,6,8};
        int i = 1;
        int n = i + v[++i] + v[++i];
        System.out.println(n);
    }
}
```

In Java, the calculated value is guaranteed to be 11.

```
$ javac Foo.java && java Foo
11
```

```c
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

"Mostly unspecified evaluation order" is a rather unique feature of C (and C++ and Fortran). Most other languages guarantees a certain evaluation order.

ASIDE

Foo.java

```java
public class Foo {
    public static void main(String args[]) {
        int v[] = {0,2,4,6,8};
        int i = 1;
        int n = i + v[++i] + v[++i];
        System.out.println(n);
    }
}
```

In Java, the calculated value is guaranteed to be 11.

```
$ javac Foo.java && java Foo
11
```

```
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

```c
#include <stdio.h>

int main(void)
{
    int v[] = {0,2,4,6,8};
    int i = 1;
    int n = i + v[++i] + v[++i];
    printf("%d\n", n);
}
```

OK, enough theory. Let's take a look under the hood and try to find out what different C compilers are actually doing here.

Let’s first change the code slightly, so it is easier to see what is happening.

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Let’s first change the code slightly, so it is easier to see what is happening.

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Let’s first change the code slightly, so it is easier to see what is happening.

The undefined behavior is still the same.

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Let’s first change the code slightly, so it is easier to see what is happening.

The undefined behavior is still the same.

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ /usr/bin/gcc main.c && ./a.out
12
$ /usr/bin/clang main.c && ./a.out
11
$ /usr/bin/icc main.c && ./a.out
13
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Let's first change the code slightly, so it is easier to see what is happening.

The undefined behavior is still the same.

Also, to make the assembler easier to read we target 32-bit architecture, tune for i386, and make sure to turn off all optimization. The observed behavior is still the same...

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Let's first change the code slightly, so it is easier to see what is happening.

The undefined behavior is still the same.

Also, to make the assembler easier to read we target 32-bit architecture, tune for i386, and make sure to turn off all optimization. The observed behavior is still the same...

On my computer (Mac OS 10.8.2, gcc 4.2.1, clang 4.1, icc 13.0.1):

```
$ /usr/bin/gcc -O0 -std=c99 -m32 -mtune=i386 main.c && ./a.out
12
$ /usr/bin/clang -O0 -std=c99 -m32 -mtune=i386 main.c && ./a.out
11
$ /usr/bin/icc -O0 -std=c99 -m32 -mtune=i386 main.c && ./a.out
13
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

I used gdb to disassemble the output, eg

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

I used gdb to disassemble the output, eg

```
$ /usr/bin/gcc -O0 -std=c99 -m32 -mtune=i386 main.c
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

I used gdb to disassemble the output, eg

```
$ /usr/bin/gcc -O0 -std=c99 -m32 -mtune=i386 main.c
$ ./a.out
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

I used gdb to disassemble the output, eg

```
$ /usr/bin/gcc -O0 -std=c99 -m32 -mtune=i386 main.c
$ ./a.out
12
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

I used gdb to disassemble the output, eg

```
$ /usr/bin/gcc -O0 -std=c99 -m32 -mtune=i386 main.c
$ ./a.out
12
$ gdb ./a.out
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

I used gdb to disassemble the output, eg

```
$ /usr/bin/gcc -O0 -std=c99 -m32 -mtune=i386 main.c
$ ./a.out
12
$ gdb ./a.out
(gdb) set disassembly-flavor intel
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

I used gdb to disassemble the output, eg

```
$ /usr/bin/gcc -O0 -std=c99 -m32 -mtune=i386 main.c
$ ./a.out
12
$ gdb ./a.out
(gdb) set disassembly-flavor intel
(gdb) disassemble main
```

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

This is the standard preamble for the function. Here it allocates space on the stack for variables with automatic storage duration (local variables).

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Here `a[0]` is loaded into a register, 1 is added and the value is written into the memory location dedicated for the local variable `i`. The variable `i` has now been initialized.

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Here `a[0]` is loaded into a register, 1 is added and the value is written into the memory location dedicated for the local variable `i`. The variable `i` has now been initialized.

Now the interesting stuff starts!

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

gcc first wants to evaluate subexpression `a[++i]`. The value stored in `i` is loaded into a register, then increased by 1, and the value is stored back to memory. Then i is loaded into register again and used to calculate the index into array a and that particular value is loaded into register. The subexpression is evaluated, and the value of this subexpresission should now be 4. The stored value of i is 2.

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Now, i is loaded into register, and added with the newly calculated value of a[++i]. 2 + 4 is 6. The result of subexpression i + a[++i] is stored in ecx and should now be 6.

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Now it is time to evaluate subexpression b[++i]. Load i into register, increase by one, store value to memory. Value of i is now 3. Load i into register again, use to calculate index into array b, and load that value into register. The value is 6.

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

The newly calculated subexpression b[++i] is now stored in edx. The old subexpression i + a[++i] is stored in ecx. They are added together.

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

The newly calculated subexpression b[++i] is now stored in edx. The old subexpression i + a[++i] is stored in ecx. They are added together.

6 + 6 is 12

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

The newly calculated subexpression b[++i] is now stored in edx. The old subexpression i + a[++i] is stored in ecx. They are added together.

6 + 6 is 12

and the result is stored into the memory location allocated for local variable n.

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

the value of n is printed

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

the value of n is printed

```
12
```

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

This is just the postamble of the main function. Store 0 as the return value, load eax with the return value. Pop the stack and return to the caller.

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

This is the interesting part of the code.

gcc

```
0x00001f10   push   ebp
0x00001f11   mov    ebp,esp
0x00001f13   sub    esp,0x18
0x00001f16   call   0x1f1b <main+11>
0x00001f1b   pop    eax
0x00001f1c   mov    ecx,DWORD PTR [eax+0x115]
0x00001f22   add    ecx,0x1
0x00001f25   mov    DWORD PTR [ebp-0xc],ecx
0x00001f28   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f2b   add    ecx,0x1
0x00001f2e   mov    DWORD PTR [ebp-0xc],ecx
0x00001f31   mov    ecx,DWORD PTR [ebp-0xc]
0x00001f34   mov    ecx,DWORD PTR [eax+ecx*4+0x115]
0x00001f3b   mov    edx,DWORD PTR [ebp-0xc]
0x00001f3e   add    ecx,edx
0x00001f40   mov    edx,DWORD PTR [ebp-0xc]
0x00001f43   add    edx,0x1
0x00001f46   mov    DWORD PTR [ebp-0xc],edx
0x00001f49   mov    edx,DWORD PTR [ebp-0xc]
0x00001f4c   mov    edx,DWORD PTR [eax+edx*4+0x135]
0x00001f53   add    ecx,edx
0x00001f55   mov    DWORD PTR [ebp-0x10],ecx
0x00001f58   mov    ecx,DWORD PTR [ebp-0x10]
0x00001f5b   mov    edx,esp
0x00001f5d   mov    DWORD PTR [edx+0x4],ecx
0x00001f60   lea    eax,[eax+0x95]
0x00001f66   mov    DWORD PTR [edx],eax
0x00001f68   call   0x1f88 <dyld_stub_printf>
0x00001f6d   mov    DWORD PTR [ebp-0x8],0x0
0x00001f74   mov    eax,DWORD PTR [ebp-0x8]
0x00001f77   mov    DWORD PTR [ebp-0x4],eax
0x00001f7a   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7d   add    esp,0x18
0x00001f80   pop    ebp
0x00001f81   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

This is the interesting part of the code.

Surprised?

Most programmers intutively think that an expression ought to be evaluated from left to right. Eg like this:

left-to-right evaluation

```
i + a[++i] + b[++i]
1 + a[++i] + b[++i]
1 + a[++1] + b[++i]
1 + a[2] + b[++i]
1 + 4 + b[++i]
5 + b[++i]
5 + b[++2]
5 + b[3]
5 + 6
11
```

Most programmers intutively think that an expression ought to be evaluated from left to right. Eg like this:

left-to-right evaluation

```
i + a[++i] + b[++i]
1 + a[++i] + b[++i]
1 + a[++1] + b[++i]
1 + a[2] + b[++i]
1 + 4 + b[++i]
5 + b[++i]
5 + b[++2]
5 + b[3]
5 + 6
11
```

Most programmers intutively think that an expression ought to be evaluated from left to right. Eg like this:

This is guaranteed to happen in most other languages, but not in C, and as we have seen, certainly not with GCC. The evaluation order of this expression is unspecified, and we are not allowed to assume anything about the sideeffects that happens when evaluating subexpressions...

left-to-right evaluation

```
i + a[++i] + b[++i]
1 + a[++i] + b[++i]
1 + a[++1] + b[++i]
1 + a[2] + b[++i]
1 + 4 + b[++i]
5 + b[++i]
5 + b[++2]
5 + b[3]
5 + 6
11
```

Most programmers intutively think that an expression ought to be evaluated from left to right. Eg like this:

This is guaranteed to happen in most other languages, but not in C, and as we have seen, certainly not with GCC. The evaluation order of this expression is unspecified, and we are not allowed to assume anything about the sideeffects that happens when evaluating subexpressions...

The evaluation we just observed looked like this:

left-to-right evaluation

```
i + a[++i] + b[++i]
1 + a[++i] + b[++i]
1 + a[++1] + b[++i]
1 + a[2] + b[++i]
1 + 4 + b[++i]
5 + b[++i]
5 + b[++2]
5 + b[3]
5 + 6
11
```

Most programmers intutively think that an expression ought to be evaluated from left to right. Eg like this:

This is guaranteed to happen in most other languages, but not in C, and as we have seen, certainly not with GCC. The evaluation order of this expression is unspecified, and we are not allowed to assume anything about the sideeffects that happens when evaluating subexpressions...

The evaluation we just observed looked like this:

gcc

```
i + a[++i] + b[++i]
i + a[++1] + b[++i]
i + a[2] + b[++i]
i + 4 + b[++i]
2 + 4 + b[++i]
6 + b[++i]
6 + b[++2]
6 + b[3]
6 + 6
12
```

left-to-right evaluation

```
i + a[++i] + b[++i]
1 + a[++i] + b[++i]
1 + a[++1] + b[++i]
1 + a[2] + b[++i]
1 + 4 + b[++i]
5 + b[++i]
5 + b[++2]
5 + b[3]
5 + 6
11
```

Most programmers intutively think that an expression ought to be evaluated from left to right. Eg like this:

This is guaranteed to happen in most other languages, but not in C, and as we have seen, certainly not with GCC. The evaluation order of this expression is unspecified, and we are not allowed to assume anything about the sideeffects that happens when evaluating subexpressions...

The evaluation we just observed looked like this:

gcc

```
i + a[++i] + b[++i]
i + a[++1] + b[++i]
i + a[2] + b[++i]
i + 4 + b[++i]
2 + 4 + b[++i]
6 + b[++i]
6 + b[++2]
6 + b[3]
6 + 6
12
```

Is gcc doing anything wrong here? Not at all. The C standard explicitly says that the evaluation order here is unspecified, and therefore gcc can do whatever it wants. The C standard also says that this is undefined behavior, so gcc could have just skipped the calculation and said value of n is 42. And still be right.

gcc

```
push   ebp
mov    ebp,esp
sub    esp,0x18
call   0x1f1b <main+11>
pop    eax
mov    ecx,DWORD PTR [eax+0x115]
add    ecx,0x1
mov    DWORD PTR [ebp-0xc],ecx
mov    ecx,DWORD PTR [ebp-0xc]
add    ecx,0x1
mov    DWORD PTR [ebp-0xc],ecx
mov    ecx,DWORD PTR [ebp-0xc]
mov    ecx,DWORD PTR [eax+ecx*4+0x115]
mov    edx,DWORD PTR [ebp-0xc]
add    ecx,edx
mov    edx,DWORD PTR [ebp-0xc]
add    edx,0x1
mov    DWORD PTR [ebp-0xc],edx
mov    edx,DWORD PTR [ebp-0xc]
mov    edx,DWORD PTR [eax+edx*4+0x135]
add    ecx,edx
mov    DWORD PTR [ebp-0x10],ecx
mov    ecx,DWORD PTR [ebp-0x10]
mov    edx,esp
mov    DWORD PTR [edx+0x4],ecx
lea    eax,[eax+0x95]
mov    DWORD PTR [edx],eax
call   0x1f88 <dyld_stub_printf>
mov    DWORD PTR [ebp-0x8],0x0
mov    eax,DWORD PTR [ebp-0x8]
mov    DWORD PTR [ebp-0x4],eax
mov    eax,DWORD PTR [ebp-0x4]
add    esp,0x18
pop    ebp
ret
```

gcc

```
push   ebp
mov    ebp,esp
sub    esp,0x18
call   0x1f1b <main+11>
pop    eax
mov    ecx,DWORD PTR [eax+0x115]
add    ecx,0x1
mov    DWORD PTR [ebp-0xc],ecx
mov    ecx,DWORD PTR [ebp-0xc]
add    ecx,0x1
mov    DWORD PTR [ebp-0xc],ecx
mov    ecx,DWORD PTR [ebp-0xc]
mov    ecx,DWORD PTR [eax+ecx*4+0x115]
mov    edx,DWORD PTR [ebp-0xc]
add    ecx,edx
mov    edx,DWORD PTR [ebp-0xc]
add    edx,0x1
mov    DWORD PTR [ebp-0xc],edx
mov    edx,DWORD PTR [ebp-0xc]
mov    edx,DWORD PTR [eax+edx*4+0x135]
add    ecx,edx
mov    DWORD PTR [ebp-0x10],ecx
mov    ecx,DWORD PTR [ebp-0x10]
mov    edx,esp
mov    DWORD PTR [edx+0x4],ecx
lea    eax,[eax+0x95]
mov    DWORD PTR [edx],eax
call   0x1f88 <dyld_stub_printf>
mov    DWORD PTR [ebp-0x8],0x0
mov    eax,DWORD PTR [ebp-0x8]
mov    DWORD PTR [ebp-0x4],eax
mov    eax,DWORD PTR [ebp-0x4]
add    esp,0x18
pop    ebp
ret
```

Here is a simple technique for writing executable psudo-assembler.

gcc

```
push    ebp
mov     ebp,esp
sub     esp,0x18
call    0x1f1b <main+11>
pop     eax
mov     ecx,DWORD PTR [eax+0x115]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
mov     ecx,DWORD PTR [eax+ecx*4+0x115]
mov     edx,DWORD PTR [ebp-0xc]
add     ecx,edx
mov     edx,DWORD PTR [ebp-0xc]
add     edx,0x1
mov     DWORD PTR [ebp-0xc],edx
mov     edx,DWORD PTR [ebp-0xc]
mov     edx,DWORD PTR [eax+edx*4+0x135]
add     ecx,edx
mov     DWORD PTR [ebp-0x10],ecx
mov     ecx,DWORD PTR [ebp-0x10]
mov     edx,esp
mov     DWORD PTR [edx+0x4],ecx
lea     eax,[eax+0x95]
mov     DWORD PTR [edx],eax
call    0x1f88 <dyld_stub_printf>
mov     DWORD PTR [ebp-0x8],0x0
mov     eax,DWORD PTR [ebp-0x8]
mov     DWORD PTR [ebp-0x4],eax
mov     eax,DWORD PTR [ebp-0x4]
add     esp,0x18
pop     ebp
ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

Here is a simple technique for writing executable psudo-assembler.

gcc

```
push    ebp
mov     ebp,esp
sub     esp,0x18
call    0x1f1b <main+11>
pop     eax
mov     ecx,DWORD PTR [eax+0x115]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
mov     ecx,DWORD PTR [eax+ecx*4+0x115]
mov     edx,DWORD PTR [ebp-0xc]
add     ecx,edx
mov     edx,DWORD PTR [ebp-0xc]
add     edx,0x1
mov     DWORD PTR [ebp-0xc],edx
mov     edx,DWORD PTR [ebp-0xc]
mov     edx,DWORD PTR [eax+edx*4+0x135]
add     ecx,edx
mov     DWORD PTR [ebp-0x10],ecx
mov     ecx,DWORD PTR [ebp-0x10]
mov     edx,esp
mov     DWORD PTR [edx+0x4],ecx
lea     eax,[eax+0x95]
mov     DWORD PTR [edx],eax
call    0x1f88 <dyld_stub_printf>
mov     DWORD PTR [ebp-0x8],0x0
mov     eax,DWORD PTR [ebp-0x8]
mov     DWORD PTR [ebp-0x4],eax
mov     eax,DWORD PTR [ebp-0x4]
add     esp,0x18
pop     ebp
ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

Here is a simple technique for writing executable psudo-assembler.

gcc

```
push    ebp
mov     ebp,esp
sub     esp,0x18
call    0x1f1b <main+11>
pop     eax
mov     ecx,DWORD PTR [eax+0x115]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
mov     ecx,DWORD PTR [eax+ecx*4+0x115]
mov     edx,DWORD PTR [ebp-0xc]
add     ecx,edx
mov     edx,DWORD PTR [ebp-0xc]
add     edx,0x1
mov     DWORD PTR [ebp-0xc],edx
mov     edx,DWORD PTR [ebp-0xc]
mov     edx,DWORD PTR [eax+edx*4+0x135]
add     ecx,edx
mov     DWORD PTR [ebp-0x10],ecx
mov     ecx,DWORD PTR [ebp-0x10]
mov     edx,esp
mov     DWORD PTR [edx+0x4],ecx
lea     eax,[eax+0x95]
mov     DWORD PTR [edx],eax
call    0x1f88 <dyld_stub_printf>
mov     DWORD PTR [ebp-0x8],0x0
mov     eax,DWORD PTR [ebp-0x8]
mov     DWORD PTR [ebp-0x4],eax
mov     eax,DWORD PTR [ebp-0x4]
add     esp,0x18
pop     ebp
ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

Here is a simple technique for writing executable psudo-assembler.

The trick is to introduce some fake registers, and then spell out the assembler in pure C.

gcc

```
push    ebp
mov     ebp,esp
sub     esp,0x18
call    0x1f1b <main+11>
pop     eax
mov     ecx,DWORD PTR [eax+0x115]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
mov     ecx,DWORD PTR [eax+ecx*4+0x115]
mov     edx,DWORD PTR [ebp-0xc]
add     ecx,edx
mov     edx,DWORD PTR [ebp-0xc]
add     edx,0x1
mov     DWORD PTR [ebp-0xc],edx
mov     edx,DWORD PTR [ebp-0xc]
mov     edx,DWORD PTR [eax+edx*4+0x135]
add     ecx,edx
mov     DWORD PTR [ebp-0x10],ecx
mov     ecx,DWORD PTR [ebp-0x10]
mov     edx,esp
mov     DWORD PTR [edx+0x4],ecx
lea     eax,[eax+0x95]
mov     DWORD PTR [edx],eax
call    0x1f88 <dyld_stub_printf>
mov     DWORD PTR [ebp-0x8],0x0
mov     eax,DWORD PTR [ebp-0x8]
mov     DWORD PTR [ebp-0x4],eax
mov     eax,DWORD PTR [ebp-0x4]
add     esp,0x18
pop     ebp
ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

gcc

```
push    ebp
mov     ebp,esp
sub     esp,0x18
call    0x1f1b <main+11>
pop     eax
mov     ecx,DWORD PTR [eax+0x115]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
mov     ecx,DWORD PTR [eax+ecx*4+0x115]
mov     edx,DWORD PTR [ebp-0xc]
add     ecx,edx
mov     edx,DWORD PTR [ebp-0xc]
add     edx,0x1
mov     DWORD PTR [ebp-0xc],edx
mov     edx,DWORD PTR [ebp-0xc]
mov     edx,DWORD PTR [eax+edx*4+0x135]
add     ecx,edx
mov     DWORD PTR [ebp-0x10],ecx
mov     ecx,DWORD PTR [ebp-0x10]
mov     edx,esp
mov     DWORD PTR [edx+0x4],ecx
lea     eax,[eax+0x95]
mov     DWORD PTR [edx],eax
call    0x1f88 <dyld_stub_printf>
mov     DWORD PTR [ebp-0x8],0x0
mov     eax,DWORD PTR [ebp-0x8]
mov     DWORD PTR [ebp-0x4],eax
mov     eax,DWORD PTR [ebp-0x4]
add     esp,0x18
pop     ebp
ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

i = a[0] + 1;

gcc

```
push    ebp
mov     ebp,esp
sub     esp,0x18
call    0x1f1b <main+11>
pop     eax
mov     ecx,DWORD PTR [eax+0x115]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
mov     ecx,DWORD PTR [eax+ecx*4+0x115]
mov     edx,DWORD PTR [ebp-0xc]
add     ecx,edx
mov     edx,DWORD PTR [ebp-0xc]
add     edx,0x1
mov     DWORD PTR [ebp-0xc],edx
mov     edx,DWORD PTR [ebp-0xc]
mov     edx,DWORD PTR [eax+edx*4+0x135]
add     ecx,edx
mov     DWORD PTR [ebp-0x10],ecx
mov     ecx,DWORD PTR [ebp-0x10]
mov     edx,esp
mov     DWORD PTR [edx+0x4],ecx
lea     eax,[eax+0x95]
mov     DWORD PTR [edx],eax
call    0x1f88 <dyld_stub_printf>
mov     DWORD PTR [ebp-0x8],0x0
mov     eax,DWORD PTR [ebp-0x8]
mov     DWORD PTR [ebp-0x4],eax
mov     eax,DWORD PTR [ebp-0x4]
add     esp,0x18
pop     ebp
ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

gcc

```
push    ebp
mov     ebp,esp
sub     esp,0x18
call    0x1f1b <main+11>
pop     eax
mov     ecx,DWORD PTR [eax+0x115]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
mov     ecx,DWORD PTR [eax+ecx*4+0x115]
mov     edx,DWORD PTR [ebp-0xc]
add     ecx,edx
mov     edx,DWORD PTR [ebp-0xc]
add     edx,0x1
mov     DWORD PTR [ebp-0xc],edx
mov     edx,DWORD PTR [ebp-0xc]
mov     edx,DWORD PTR [eax+edx*4+0x135]
add     ecx,edx
mov     DWORD PTR [ebp-0x10],ecx
mov     ecx,DWORD PTR [ebp-0x10]
mov     edx,esp
mov     DWORD PTR [edx+0x4],ecx
lea     eax,[eax+0x95]
mov     DWORD PTR [edx],eax
call    0x1f88 <dyld_stub_printf>
mov     DWORD PTR [ebp-0x8],0x0
mov     eax,DWORD PTR [ebp-0x8]
mov     DWORD PTR [ebp-0x4],eax
mov     eax,DWORD PTR [ebp-0x4]
add     esp,0x18
pop     ebp
ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

a[++i]

gcc

```
push   ebp
mov    ebp,esp
sub    esp,0x18
call   0x1f1b <main+11>
pop    eax
mov    ecx,DWORD PTR [eax+0x115]
add    ecx,0x1
mov    DWORD PTR [ebp-0xc],ecx
mov    ecx,DWORD PTR [ebp-0xc]
add    ecx,0x1
mov    DWORD PTR [ebp-0xc],ecx
mov    ecx,DWORD PTR [ebp-0xc]
mov    ecx,DWORD PTR [eax+ecx*4+0x115]
mov    edx,DWORD PTR [ebp-0xc]
add    ecx,edx
mov    edx,DWORD PTR [ebp-0xc]
add    edx,0x1
mov    DWORD PTR [ebp-0xc],edx
mov    edx,DWORD PTR [ebp-0xc]
mov    edx,DWORD PTR [eax+edx*4+0x135]
add    ecx,edx
mov    DWORD PTR [ebp-0x10],ecx
mov    ecx,DWORD PTR [ebp-0x10]
mov    edx,esp
mov    DWORD PTR [edx+0x4],ecx
lea    eax,[eax+0x95]
mov    DWORD PTR [edx],eax
call   0x1f88 <dyld_stub_printf>
mov    DWORD PTR [ebp-0x8],0x0
mov    eax,DWORD PTR [ebp-0x8]
mov    DWORD PTR [ebp-0x4],eax
mov    eax,DWORD PTR [ebp-0x4]
add    esp,0x18
pop    ebp
ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

gcc

```
push    ebp
mov     ebp,esp
sub     esp,0x18
call    0x1f1b <main+11>
pop     eax
mov     ecx,DWORD PTR [eax+0x115]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
mov     ecx,DWORD PTR [eax+ecx*4+0x115]
mov     edx,DWORD PTR [ebp-0xc]
add     ecx,edx
mov     edx,DWORD PTR [ebp-0xc]
add     edx,0x1
mov     DWORD PTR [ebp-0xc],edx
mov     edx,DWORD PTR [ebp-0xc]
mov     edx,DWORD PTR [eax+edx*4+0x135]
add     ecx,edx
mov     DWORD PTR [ebp-0x10],ecx
mov     ecx,DWORD PTR [ebp-0x10]
mov     edx,esp
mov     DWORD PTR [edx+0x4],ecx
lea     eax,[eax+0x95]
mov     DWORD PTR [edx],eax
call    0x1f88 <dyld_stub_printf>
mov     DWORD PTR [ebp-0x8],0x0
mov     eax,DWORD PTR [ebp-0x8]
mov     DWORD PTR [ebp-0x4],eax
mov     eax,DWORD PTR [ebp-0x4]
add     esp,0x18
pop     ebp
ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

i + (a[++i])

gcc

```
push    ebp
mov     ebp,esp
sub     esp,0x18
call    0x1f1b <main+11>
pop     eax
mov     ecx,DWORD PTR [eax+0x115]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
mov     ecx,DWORD PTR [eax+ecx*4+0x115]
mov     edx,DWORD PTR [ebp-0xc]
add     ecx,edx
mov     edx,DWORD PTR [ebp-0xc]
add     edx,0x1
mov     DWORD PTR [ebp-0xc],edx
mov     edx,DWORD PTR [ebp-0xc]
mov     edx,DWORD PTR [eax+edx*4+0x135]
add     ecx,edx
mov     DWORD PTR [ebp-0x10],ecx
mov     ecx,DWORD PTR [ebp-0x10]
mov     edx,esp
mov     DWORD PTR [edx+0x4],ecx
lea     eax,[eax+0x95]
mov     DWORD PTR [edx],eax
call    0x1f88 <dyld_stub_printf>
mov     DWORD PTR [ebp-0x8],0x0
mov     eax,DWORD PTR [ebp-0x8]
mov     DWORD PTR [ebp-0x4],eax
mov     eax,DWORD PTR [ebp-0x4]
add     esp,0x18
pop     ebp
ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

gcc

```
push   ebp
mov    ebp,esp
sub    esp,0x18
call   0x1f1b <main+11>
pop    eax
mov    ecx,DWORD PTR [eax+0x115]
add    ecx,0x1
mov    DWORD PTR [ebp-0xc],ecx
mov    ecx,DWORD PTR [ebp-0xc]
add    ecx,0x1
mov    DWORD PTR [ebp-0xc],ecx
mov    ecx,DWORD PTR [ebp-0xc]
mov    ecx,DWORD PTR [eax+ecx*4+0x115]
mov    edx,DWORD PTR [ebp-0xc]
add    ecx,edx
mov    edx,DWORD PTR [ebp-0xc]
add    edx,0x1
mov    DWORD PTR [ebp-0xc],edx
mov    edx,DWORD PTR [ebp-0xc]
mov    edx,DWORD PTR [eax+edx*4+0x135]
add    ecx,edx
mov    DWORD PTR [ebp-0x10],ecx
mov    ecx,DWORD PTR [ebp-0x10]
mov    edx,esp
mov    DWORD PTR [edx+0x4],ecx
lea    eax,[eax+0x95]
mov    DWORD PTR [edx],eax
call   0x1f88 <dyld_stub_printf>
mov    DWORD PTR [ebp-0x8],0x0
mov    eax,DWORD PTR [ebp-0x8]
mov    DWORD PTR [ebp-0x4],eax
mov    eax,DWORD PTR [ebp-0x4]
add    esp,0x18
pop    ebp
ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

b[++i]

gcc

```
push   ebp
mov    ebp,esp
sub    esp,0x18
call   0x1f1b <main+11>
pop    eax
mov    ecx,DWORD PTR [eax+0x115]
add    ecx,0x1
mov    DWORD PTR [ebp-0xc],ecx
mov    ecx,DWORD PTR [ebp-0xc]
add    ecx,0x1
mov    DWORD PTR [ebp-0xc],ecx
mov    ecx,DWORD PTR [ebp-0xc]
mov    ecx,DWORD PTR [eax+ecx*4+0x115]
mov    edx,DWORD PTR [ebp-0xc]
add    ecx,edx
mov    edx,DWORD PTR [ebp-0xc]
add    edx,0x1
mov    DWORD PTR [ebp-0xc],edx
mov    edx,DWORD PTR [ebp-0xc]
mov    edx,DWORD PTR [eax+edx*4+0x135]
add    ecx,edx
mov    DWORD PTR [ebp-0x10],ecx
mov    ecx,DWORD PTR [ebp-0x10]
mov    edx,esp
mov    DWORD PTR [edx+0x4],ecx
lea    eax,[eax+0x95]
mov    DWORD PTR [edx],eax
call   0x1f88 <dyld_stub_printf>
mov    DWORD PTR [ebp-0x8],0x0
mov    eax,DWORD PTR [ebp-0x8]
mov    DWORD PTR [ebp-0x4],eax
mov    eax,DWORD PTR [ebp-0x4]
add    esp,0x18
pop    ebp
ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

gcc

```
push    ebp
mov     ebp,esp
sub     esp,0x18
call    0x1f1b <main+11>
pop     eax
mov     ecx,DWORD PTR [eax+0x115]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
mov     ecx,DWORD PTR [eax+ecx*4+0x115]
mov     edx,DWORD PTR [ebp-0xc]
add     ecx,edx
mov     edx,DWORD PTR [ebp-0xc]
add     edx,0x1
mov     DWORD PTR [ebp-0xc],edx
mov     edx,DWORD PTR [ebp-0xc]
mov     edx,DWORD PTR [eax+edx*4+0x135]
add     ecx,edx
mov     DWORD PTR [ebp-0x10],ecx
mov     ecx,DWORD PTR [ebp-0x10]
mov     edx,esp
mov     DWORD PTR [edx+0x4],ecx
lea     eax,[eax+0x95]
mov     DWORD PTR [edx],eax
call    0x1f88 <dyld_stub_printf>
mov     DWORD PTR [ebp-0x8],0x0
mov     eax,DWORD PTR [ebp-0x8]
mov     DWORD PTR [ebp-0x4],eax
mov     eax,DWORD PTR [ebp-0x4]
add     esp,0x18
pop     ebp
ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

```
n = (i + a[++i]) +
    (b[++i])
```

gcc

```
push    ebp
mov     ebp,esp
sub     esp,0x18
call    0x1f1b <main+11>
pop     eax
mov     ecx,DWORD PTR [eax+0x115]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
mov     ecx,DWORD PTR [eax+ecx*4+0x115]
mov     edx,DWORD PTR [ebp-0xc]
add     ecx,edx
mov     edx,DWORD PTR [ebp-0xc]
add     edx,0x1
mov     DWORD PTR [ebp-0xc],edx
mov     edx,DWORD PTR [ebp-0xc]
mov     edx,DWORD PTR [eax+edx*4+0x135]
add     ecx,edx
mov     DWORD PTR [ebp-0x10],ecx
mov     ecx,DWORD PTR [ebp-0x10]
mov     edx,esp
mov     DWORD PTR [edx+0x4],ecx
lea     eax,[eax+0x95]
mov     DWORD PTR [edx],eax
call    0x1f88 <dyld_stub_printf>
mov     DWORD PTR [ebp-0x8],0x0
mov     eax,DWORD PTR [ebp-0x8]
mov     DWORD PTR [ebp-0x4],eax
mov     eax,DWORD PTR [ebp-0x4]
add     esp,0x18
pop     ebp
ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

gcc

```
push    ebp
mov     ebp,esp
sub     esp,0x18
call    0x1f1b <main+11>
pop     eax
mov     ecx,DWORD PTR [eax+0x115]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
add     ecx,0x1
mov     DWORD PTR [ebp-0xc],ecx
mov     ecx,DWORD PTR [ebp-0xc]
mov     ecx,DWORD PTR [eax+ecx*4+0x115]
mov     edx,DWORD PTR [ebp-0xc]
add     ecx,edx
mov     edx,DWORD PTR [ebp-0xc]
add     edx,0x1
mov     DWORD PTR [ebp-0xc],edx
mov     edx,DWORD PTR [ebp-0xc]
mov     edx,DWORD PTR [eax+edx*4+0x135]
add     ecx,edx
mov     DWORD PTR [ebp-0x10],ecx
mov     ecx,DWORD PTR [ebp-0x10]
mov     edx,esp
mov     DWORD PTR [edx+0x4],ecx
lea     eax,[eax+0x95]
mov     DWORD PTR [edx],eax
call    0x1f88 <dyld_stub_printf>
mov     DWORD PTR [ebp-0x8],0x0
mov     eax,DWORD PTR [ebp-0x8]
mov     DWORD PTR [ebp-0x4],eax
mov     eax,DWORD PTR [ebp-0x4]
add     esp,0x18
pop     ebp
ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

```
12
```

You can take this idea of writing executable pseudo-assembler even further.

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

You can take this idea of writing executable pseudo-assembler even further.

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {
    int x,y,z;
} tmp;

int main(void) {
    int i, n;

    i = a[0] + 1;

    ++i;
    tmp.x = a[i];

    tmp.x += i;

    ++i;
    tmp.y = b[i];

    tmp.x += tmp.y;
    n = tmp.x;

    printf("%d\n", n);
}
```

You can take this idea of writing executable pseudo-assembler even further.

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {
    int x,y,z;
} tmp;

int main(void) {
    int i, n;

    i = a[0] + 1;

    ++i;
    tmp.x = a[i];

    tmp.x += i;

    ++i;
    tmp.y = b[i];

    tmp.x += tmp.y;
    n = tmp.x;

    printf("%d\n", n);
}
```

You can take this idea of writing executable pseudo-assembler even further.

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {
    int x,y,z;
} tmp;

int main(void) {
    int i, n;

    i = a[0] + 1;

    ++i;
    tmp.x = a[i];

    tmp.x += i;

    ++i;
    tmp.y = b[i];

    tmp.x += tmp.y;
    n = tmp.x;

    printf("%d\n", n);
}
```

You can take this idea of writing executable pseudo-assembler even further.

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {
    int x,y,z;
} tmp;

int main(void) {
    int i, n;

    i = a[0] + 1;

    ++i;
    tmp.x = a[i];

    tmp.x += i;

    ++i;
    tmp.y = b[i];

    tmp.x += tmp.y;
    n = tmp.x;

    printf("%d\n", n);
}
```

You can take this idea of writing executable pseudo-assembler even further.

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {
    int x,y,z;
} tmp;

int main(void) {
    int i, n;

    i = a[0] + 1;

    ++i;
    tmp.x = a[i];

    tmp.x += i;

    ++i;
    tmp.y = b[i];

    tmp.x += tmp.y;
    n = tmp.x;

    printf("%d\n", n);
}
```

You can take this idea of writing executable pseudo-assembler even further.

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {
    int x,y,z;
} tmp;

int main(void) {
    int i, n;

    i = a[0] + 1;

    ++i;
    tmp.x = a[i];

    tmp.x += i;

    ++i;
    tmp.y = b[i];

    tmp.x += tmp.y;
    n = tmp.x;

    printf("%d\n", n);
}
```

You can take this idea of writing executable pseudo-assembler even further.

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {
    int x,y,z;
} tmp;

int main(void) {
    int i, n;

    i = a[0] + 1;

    ++i;
    tmp.x = a[i];

    tmp.x += i;

    ++i;
    tmp.y = b[i];

    tmp.x += tmp.y;
    n = tmp.x;

    printf("%d\n", n);
}
```

You can take this idea of writing executable pseudo-assembler even further.

You can take this idea of writing executable pseudo-assembler even further.

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {
    int x,y,z;
} tmp;

int main(void) {
    int i, n;

    i = a[0] + 1;

    ++i;
    tmp.x = a[i];

    tmp.x += i;

    ++i;
    tmp.y = b[i];

    tmp.x += tmp.y;
    n = tmp.x;

    printf("%d\n", n);
}
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {
    int x,y,z;
} tmp;

int main(void) {
    int i, n;

    i = a[0] + 1;

    // n = i + a[++i] + b[++i]
    tmp.x = a[++i];
    tmp.x += i;
    tmp.y = b[++i];
    n = tmp.x + tmp.y;

    printf("%d\n", n);
}
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct reg {
    int a,b,c,d;
} reg;

int main(void) {
    int i, n;

    reg.c = a[0];
    reg.c += 1;
    i = reg.c;

    reg.c = i;
    reg.c += 1;
    i = reg.c;
    reg.c = i;
    reg.c = a[reg.c];

    reg.d = i;
    reg.c += reg.d;

    reg.d = i;
    reg.d += 1;
    i = reg.d;
    reg.d = i;
    reg.d = b[reg.d];

    reg.c += reg.d;
    n = reg.c;

    printf("%d\n", n);
}
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {
    int x,y,z;
} tmp;

int main(void) {
    int i, n;

    i = a[0] + 1;

    ++i;
    tmp.x = a[i];

    tmp.x += i;

    ++i;
    tmp.y = b[i];

    tmp.x += tmp.y;
    n = tmp.x;

    printf("%d\n", n);
}
```

You can take this idea of writing executable pseudo-assembler even further.

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {
    int x,y,z;
} tmp;

int main(void) {
    int i, n;

    i = a[0] + 1;

    // n = i + a[++i] + b[++i]
    tmp.x = a[++i];
    tmp.x += i;
    tmp.y = b[++i];
    n = tmp.x + tmp.y;

    printf("%d\n", n);
}
```

```
12
```

```
n = i + a[++i] + b[++i];
```

```
n = i + a[++i] + b[++i];
```

We have just seen how gcc "interprets" this (meaningless) expression.

```
n = i + a[++i] + b[++i];
```

gcc

```
tmp.x = a[++i];
tmp.x += i;
tmp.y = b[++i];
n = tmp.x + tmp.y;
```

We have just seen how gcc "interprets" this (meaningless) expression.

```
n = i + a[++i] + b[++i];
```

We have just seen how gcc "interprets" this (meaningless) expression.

gcc

```
tmp.x = a[++i];
tmp.x += i;
tmp.y = b[++i];
n = tmp.x + tmp.y;
```

gcc

```
i + a[++i] + b[++i]
i + a[++1] + b[++i]
i + a[2] + b[++i]
i + 4 + b[++i]
2 + 4 + b[++i]
6 + b[++i]
6 + b[++2]
6 + b[3]
6 + 6
12
```

icc

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

icc

```
0x00001f54    push    ebp
0x00001f55    mov     ebp,esp
0x00001f57    sub     esp,0x28
0x00001f5a    mov     DWORD PTR [ebp-0x10],ebx
0x00001f5d    call    0x1f62 <main+14>
0x00001f62    pop     eax
0x00001f63    lea     edx,[eax+0xc2]
0x00001f69    mov     ecx,0x1
0x00001f6e    add     ecx,DWORD PTR [edx]
0x00001f70    mov     DWORD PTR [ebp-0x18],ecx
0x00001f73    mov     edx,0x1
0x00001f78    add     edx,DWORD PTR [ebp-0x18]
0x00001f7b    mov     DWORD PTR [ebp-0x18],edx
0x00001f7e    mov     ecx,0x1
0x00001f83    add     ecx,DWORD PTR [ebp-0x18]
0x00001f86    mov     DWORD PTR [ebp-0x18],ecx
0x00001f89    shl     edx,0x2
0x00001f8c    lea     ebx,[eax+0xc2]
0x00001f92    add     ebx,edx
0x00001f94    mov     edx,DWORD PTR [ebx]
0x00001f96    add     edx,DWORD PTR [ebp-0x18]
0x00001f99    shl     ecx,0x2
0x00001f9c    lea     ebx,[eax+0xd6]
0x00001fa2    add     ebx,ecx
0x00001fa4    add     edx,DWORD PTR [ebx]
0x00001fa6    mov     DWORD PTR [ebp-0x14],edx
0x00001fa9    lea     eax,[eax+0x9a]
0x00001faf    mov     DWORD PTR [esp],eax
0x00001fb2    mov     eax,DWORD PTR [ebp-0x14]
0x00001fb5    mov     DWORD PTR [esp+0x4],eax
0x00001fb9    call    0x1fd6 <dyld_stub_printf>
0x00001fbe    add     esp,0x10
0x00001fc1    mov     eax,0x0
0x00001fc6    mov     ebx,DWORD PTR [ebp-0x10]
0x00001fc9    leave
0x00001fca    ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

The preamble

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

icc

```
0x00001f54    push   ebp
0x00001f55    mov    ebp,esp
0x00001f57    sub    esp,0x28
0x00001f5a    mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d    call   0x1f62 <main+14>
0x00001f62    pop    eax
0x00001f63    lea    edx,[eax+0xc2]
0x00001f69    mov    ecx,0x1
0x00001f6e    add    ecx,DWORD PTR [edx]
0x00001f70    mov    DWORD PTR [ebp-0x18],ecx
0x00001f73    mov    edx,0x1
0x00001f78    add    edx,DWORD PTR [ebp-0x18]
0x00001f7b    mov    DWORD PTR [ebp-0x18],edx
0x00001f7e    mov    ecx,0x1
0x00001f83    add    ecx,DWORD PTR [ebp-0x18]
0x00001f86    mov    DWORD PTR [ebp-0x18],ecx
0x00001f89    shl    edx,0x2
0x00001f8c    lea    ebx,[eax+0xc2]
0x00001f92    add    ebx,edx
0x00001f94    mov    edx,DWORD PTR [ebx]
0x00001f96    add    edx,DWORD PTR [ebp-0x18]
0x00001f99    shl    ecx,0x2
0x00001f9c    lea    ebx,[eax+0xd6]
0x00001fa2    add    ebx,ecx
0x00001fa4    add    edx,DWORD PTR [ebx]
0x00001fa6    mov    DWORD PTR [ebp-0x14],edx
0x00001fa9    lea    eax,[eax+0x9a]
0x00001faf    mov    DWORD PTR [esp],eax
0x00001fb2    mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5    mov    DWORD PTR [esp+0x4],eax
0x00001fb9    call   0x1fd6 <dyld_stub_printf>
0x00001fbe    add    esp,0x10
0x00001fc1    mov    eax,0x0
0x00001fc6    mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9    leave
0x00001fca    ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

initialize i to a[0] + 1 = 1

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

increase the stored value of i to 2.
Keep the value 2 in edx.

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

increase the stored value of i to 3.
Keep the value 3 in ecx.

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

icc

```
0x00001f54    push   ebp
0x00001f55    mov    ebp,esp
0x00001f57    sub    esp,0x28
0x00001f5a    mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d    call   0x1f62 <main+14>
0x00001f62    pop    eax
0x00001f63    lea    edx,[eax+0xc2]
0x00001f69    mov    ecx,0x1
0x00001f6e    add    ecx,DWORD PTR [edx]
0x00001f70    mov    DWORD PTR [ebp-0x18],ecx
0x00001f73    mov    edx,0x1
0x00001f78    add    edx,DWORD PTR [ebp-0x18]
0x00001f7b    mov    DWORD PTR [ebp-0x18],edx
0x00001f7e    mov    ecx,0x1
0x00001f83    add    ecx,DWORD PTR [ebp-0x18]
0x00001f86    mov    DWORD PTR [ebp-0x18],ecx
0x00001f89    shl    edx,0x2
0x00001f8c    lea    ebx,[eax+0xc2]
0x00001f92    add    ebx,edx
0x00001f94    mov    edx,DWORD PTR [ebx]
0x00001f96    add    edx,DWORD PTR [ebp-0x18]
0x00001f99    shl    ecx,0x2
0x00001f9c    lea    ebx,[eax+0xd6]
0x00001fa2    add    ebx,ecx
0x00001fa4    add    edx,DWORD PTR [ebx]
0x00001fa6    mov    DWORD PTR [ebp-0x14],edx
0x00001fa9    lea    eax,[eax+0x9a]
0x00001faf    mov    DWORD PTR [esp],eax
0x00001fb2    mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5    mov    DWORD PTR [esp+0x4],eax
0x00001fb9    call   0x1fd6 <dyld_stub_printf>
0x00001fbe    add    esp,0x10
0x00001fc1    mov    eax,0x0
0x00001fc6    mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9    leave
0x00001fca    ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

It looks like icc is first "scanning" through the expression, applying the side effects, and then compute a result.

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

icc

```
0x00001f54    push   ebp
0x00001f55    mov    ebp,esp
0x00001f57    sub    esp,0x28
0x00001f5a    mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d    call   0x1f62 <main+14>
0x00001f62    pop    eax
0x00001f63    lea    edx,[eax+0xc2]
0x00001f69    mov    ecx,0x1
0x00001f6e    add    ecx,DWORD PTR [edx]
0x00001f70    mov    DWORD PTR [ebp-0x18],ecx
0x00001f73    mov    edx,0x1
0x00001f78    add    edx,DWORD PTR [ebp-0x18]
0x00001f7b    mov    DWORD PTR [ebp-0x18],edx
0x00001f7e    mov    ecx,0x1
0x00001f83    add    ecx,DWORD PTR [ebp-0x18]
0x00001f86    mov    DWORD PTR [ebp-0x18],ecx
0x00001f89    shl    edx,0x2
0x00001f8c    lea    ebx,[eax+0xc2]
0x00001f92    add    ebx,edx
0x00001f94    mov    edx,DWORD PTR [ebx]
0x00001f96    add    edx,DWORD PTR [ebp-0x18]
0x00001f99    shl    ecx,0x2
0x00001f9c    lea    ebx,[eax+0xd6]
0x00001fa2    add    ebx,ecx
0x00001fa4    add    edx,DWORD PTR [ebx]
0x00001fa6    mov    DWORD PTR [ebp-0x14],edx
0x00001fa9    lea    eax,[eax+0x9a]
0x00001faf    mov    DWORD PTR [esp],eax
0x00001fb2    mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5    mov    DWORD PTR [esp+0x4],eax
0x00001fb9    call   0x1fd6 <dyld_stub_printf>
0x00001fbe    add    esp,0x10
0x00001fc1    mov    eax,0x0
0x00001fc6    mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9    leave
0x00001fca    ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

edx is 2, multiply by 4 (sizeof int), use it to index from into array a. Load the value of a[2] into edx. This is just a fancy way of indexing into an array. edx is now 4.

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

add the stored value of i with the evaluated value of a[++i]. Notice that i is 3 because it has been updated twice already.

icc

```
0x00001f54    push   ebp
0x00001f55    mov    ebp,esp
0x00001f57    sub    esp,0x28
0x00001f5a    mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d    call   0x1f62 <main+14>
0x00001f62    pop    eax
0x00001f63    lea    edx,[eax+0xc2]
0x00001f69    mov    ecx,0x1
0x00001f6e    add    ecx,DWORD PTR [edx]
0x00001f70    mov    DWORD PTR [ebp-0x18],ecx
0x00001f73    mov    edx,0x1
0x00001f78    add    edx,DWORD PTR [ebp-0x18]
0x00001f7b    mov    DWORD PTR [ebp-0x18],edx
0x00001f7e    mov    ecx,0x1
0x00001f83    add    ecx,DWORD PTR [ebp-0x18]
0x00001f86    mov    DWORD PTR [ebp-0x18],ecx
0x00001f89    shl    edx,0x2
0x00001f8c    lea    ebx,[eax+0xc2]
0x00001f92    add    ebx,edx
0x00001f94    mov    edx,DWORD PTR [ebx]
0x00001f96    add    edx,DWORD PTR [ebp-0x18]
0x00001f99    shl    ecx,0x2
0x00001f9c    lea    ebx,[eax+0xd6]
0x00001fa2    add    ebx,ecx
0x00001fa4    add    edx,DWORD PTR [ebx]
0x00001fa6    mov    DWORD PTR [ebp-0x14],edx
0x00001fa9    lea    eax,[eax+0x9a]
0x00001faf    mov    DWORD PTR [esp],eax
0x00001fb2    mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5    mov    DWORD PTR [esp+0x4],eax
0x00001fb9    call   0x1fd6 <dyld_stub_printf>
0x00001fbe    add    esp,0x10
0x00001fc1    mov    eax,0x0
0x00001fc6    mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9    leave
0x00001fca    ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

add the stored value of i with the evaluated value of a[++i]. Notice that i is 3 because it has been updated twice already.

3 + 4 = 7

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

icc

```
0x00001f54    push   ebp
0x00001f55    mov    ebp,esp
0x00001f57    sub    esp,0x28
0x00001f5a    mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d    call   0x1f62 <main+14>
0x00001f62    pop    eax
0x00001f63    lea    edx,[eax+0xc2]
0x00001f69    mov    ecx,0x1
0x00001f6e    add    ecx,DWORD PTR [edx]
0x00001f70    mov    DWORD PTR [ebp-0x18],ecx
0x00001f73    mov    edx,0x1
0x00001f78    add    edx,DWORD PTR [ebp-0x18]
0x00001f7b    mov    DWORD PTR [ebp-0x18],edx
0x00001f7e    mov    ecx,0x1
0x00001f83    add    ecx,DWORD PTR [ebp-0x18]
0x00001f86    mov    DWORD PTR [ebp-0x18],ecx
0x00001f89    shl    edx,0x2
0x00001f8c    lea    ebx,[eax+0xc2]
0x00001f92    add    ebx,edx
0x00001f94    mov    edx,DWORD PTR [ebx]
0x00001f96    add    edx,DWORD PTR [ebp-0x18]
0x00001f99    shl    ecx,0x2
0x00001f9c    lea    ebx,[eax+0xd6]
0x00001fa2    add    ebx,ecx
0x00001fa4    add    edx,DWORD PTR [ebx]
0x00001fa6    mov    DWORD PTR [ebp-0x14],edx
0x00001fa9    lea    eax,[eax+0x9a]
0x00001faf    mov    DWORD PTR [esp],eax
0x00001fb2    mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5    mov    DWORD PTR [esp+0x4],eax
0x00001fb9    call   0x1fd6 <dyld_stub_printf>
0x00001fbe    add    esp,0x10
0x00001fc1    mov    eax,0x0
0x00001fc6    mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9    leave
0x00001fca    ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

ecx is 3, multiply by 4 (sizeof int), use it to index from into array b. Load the address of b[3] into ebx.

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

add the value of b[3] to edx.
7 + 6 = 13

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Initialize n to 13.

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Print out and exit.

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Print out and exit.

```
13
```

icc

```
0x00001f54   push   ebp
0x00001f55   mov    ebp,esp
0x00001f57   sub    esp,0x28
0x00001f5a   mov    DWORD PTR [ebp-0x10],ebx
0x00001f5d   call   0x1f62 <main+14>
0x00001f62   pop    eax
0x00001f63   lea    edx,[eax+0xc2]
0x00001f69   mov    ecx,0x1
0x00001f6e   add    ecx,DWORD PTR [edx]
0x00001f70   mov    DWORD PTR [ebp-0x18],ecx
0x00001f73   mov    edx,0x1
0x00001f78   add    edx,DWORD PTR [ebp-0x18]
0x00001f7b   mov    DWORD PTR [ebp-0x18],edx
0x00001f7e   mov    ecx,0x1
0x00001f83   add    ecx,DWORD PTR [ebp-0x18]
0x00001f86   mov    DWORD PTR [ebp-0x18],ecx
0x00001f89   shl    edx,0x2
0x00001f8c   lea    ebx,[eax+0xc2]
0x00001f92   add    ebx,edx
0x00001f94   mov    edx,DWORD PTR [ebx]
0x00001f96   add    edx,DWORD PTR [ebp-0x18]
0x00001f99   shl    ecx,0x2
0x00001f9c   lea    ebx,[eax+0xd6]
0x00001fa2   add    ebx,ecx
0x00001fa4   add    edx,DWORD PTR [ebx]
0x00001fa6   mov    DWORD PTR [ebp-0x14],edx
0x00001fa9   lea    eax,[eax+0x9a]
0x00001faf   mov    DWORD PTR [esp],eax
0x00001fb2   mov    eax,DWORD PTR [ebp-0x14]
0x00001fb5   mov    DWORD PTR [esp+0x4],eax
0x00001fb9   call   0x1fd6 <dyld_stub_printf>
0x00001fbe   add    esp,0x10
0x00001fc1   mov    eax,0x0
0x00001fc6   mov    ebx,DWORD PTR [ebp-0x10]
0x00001fc9   leave
0x00001fca   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {int x,y,z;} tmp;

int main(void)
{
    int i;
    int n;

    i = a[0] + 1;
    tmp.x = ++i;
    tmp.y = ++i;
    tmp.z = a[tmp.x];
    tmp.z += i;
    tmp.z += b[tmp.y];
    n = tmp.z;

    printf("%d\n", n);
}
```

```
n = i + a[++i] + b[++i];
```

```
n = i + a[++i] + b[++i];
```

This is how icc interprets this expression

```
n = i + a[++i] + b[++i];
```

icc

```
i = a[0] + 1;
tmp.x = ++i;
tmp.y = ++i;
tmp.z = a[tmp.x];
tmp.z += i;
tmp.z += b[tmp.y];
n = tmp.z;
```

This is how icc interprets this expression

```
n = i + a[++i] + b[++i];
```

icc

```
i = a[0] + 1;
tmp.x = ++i;
tmp.y = ++i;
tmp.z = a[tmp.x];
tmp.z += i;
tmp.z += b[tmp.y];
n = tmp.z;
```

This is how icc interprets this expression

icc

```
i + a[++i] + b[++i]
i + a[++1] + b[++i]
i + a[2] + b[++i]
i + a[2] + b[++2]
i + a[2] + b[3]
3 + 4 + b[3]
7 + b[3]
7 + 6
13
```

clang

clang

```
0x00001f20    push   ebp
0x00001f21    mov    ebp,esp
0x00001f23    push   edi
0x00001f24    push   esi
0x00001f25    sub    esp,0x20
0x00001f28    call   0x1f2d <main+13>
0x00001f2d    pop    eax
0x00001f2e    lea    ecx,[eax+0x85]
0x00001f34    mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a    add    edx,0x1
0x00001f40    mov    DWORD PTR [ebp-0xc],edx
0x00001f43    mov    edx,DWORD PTR [ebp-0xc]
0x00001f46    mov    esi,DWORD PTR [ebp-0xc]
0x00001f49    mov    edi,esi
0x00001f4b    add    edi,0x1
0x00001f51    mov    DWORD PTR [ebp-0xc],edi
0x00001f54    add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b    mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e    mov    edi,esi
0x00001f60    add    edi,0x1
0x00001f66    mov    DWORD PTR [ebp-0xc],edi
0x00001f69    add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70    mov    DWORD PTR [ebp-0x10],edx
0x00001f73    mov    eax,DWORD PTR [ebp-0x10]
0x00001f76    mov    DWORD PTR [esp],ecx
0x00001f79    mov    DWORD PTR [esp+0x4],eax
0x00001f7d    call   0x1f94 <dyld_stub_printf>
0x00001f82    mov    ecx,0x0
0x00001f87    mov    DWORD PTR [ebp-0x14],eax
0x00001f8a    mov    eax,ecx
0x00001f8c    add    esp,0x20
0x00001f8f    pop    esi
0x00001f90    pop    edi
0x00001f91    pop    ebp
0x00001f92    ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

The preamble

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

load pointer to string literal into register

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Initialize i

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Evaluate i, save value in register edx

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

load value of i into register.

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

increase the stored value of i

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

Load a[2] into register. Note how the old value of i is used to index into array a, it seems like clang indexes from &a[1]. Anyway value of edx was 1. 4 is added. New value is 5.

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

load i into register, increase and store.

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

clang

```
0x00001f20    push    ebp
0x00001f21    mov     ebp,esp
0x00001f23    push    edi
0x00001f24    push    esi
0x00001f25    sub     esp,0x20
0x00001f28    call    0x1f2d <main+13>
0x00001f2d    pop     eax
0x00001f2e    lea     ecx,[eax+0x85]
0x00001f34    mov     edx,DWORD PTR [eax+0xdf]
0x00001f3a    add     edx,0x1
0x00001f40    mov     DWORD PTR [ebp-0xc],edx
0x00001f43    mov     edx,DWORD PTR [ebp-0xc]
0x00001f46    mov     esi,DWORD PTR [ebp-0xc]
0x00001f49    mov     edi,esi
0x00001f4b    add     edi,0x1
0x00001f51    mov     DWORD PTR [ebp-0xc],edi
0x00001f54    add     edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b    mov     esi,DWORD PTR [ebp-0xc]
0x00001f5e    mov     edi,esi
0x00001f60    add     edi,0x1
0x00001f66    mov     DWORD PTR [ebp-0xc],edi
0x00001f69    add     edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70    mov     DWORD PTR [ebp-0x10],edx
0x00001f73    mov     eax,DWORD PTR [ebp-0x10]
0x00001f76    mov     DWORD PTR [esp],ecx
0x00001f79    mov     DWORD PTR [esp+0x4],eax
0x00001f7d    call    0x1f94 <dyld_stub_printf>
0x00001f82    mov     ecx,0x0
0x00001f87    mov     DWORD PTR [ebp-0x14],eax
0x00001f8a    mov     eax,ecx
0x00001f8c    add     esp,0x20
0x00001f8f    pop     esi
0x00001f90    pop     edi
0x00001f91    pop     ebp
0x00001f92    ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

edx was 5, now add b[3], and edx is now 11.

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

initialize n to 11

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

print the value

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

print the value

```
11
```

clang

```
0x00001f20   push    ebp
0x00001f21   mov     ebp,esp
0x00001f23   push    edi
0x00001f24   push    esi
0x00001f25   sub     esp,0x20
0x00001f28   call    0x1f2d <main+13>
0x00001f2d   pop     eax
0x00001f2e   lea     ecx,[eax+0x85]
0x00001f34   mov     edx,DWORD PTR [eax+0xdf]
0x00001f3a   add     edx,0x1
0x00001f40   mov     DWORD PTR [ebp-0xc],edx
0x00001f43   mov     edx,DWORD PTR [ebp-0xc]
0x00001f46   mov     esi,DWORD PTR [ebp-0xc]
0x00001f49   mov     edi,esi
0x00001f4b   add     edi,0x1
0x00001f51   mov     DWORD PTR [ebp-0xc],edi
0x00001f54   add     edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov     esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov     edi,esi
0x00001f60   add     edi,0x1
0x00001f66   mov     DWORD PTR [ebp-0xc],edi
0x00001f69   add     edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov     DWORD PTR [ebp-0x10],edx
0x00001f73   mov     eax,DWORD PTR [ebp-0x10]
0x00001f76   mov     DWORD PTR [esp],ecx
0x00001f79   mov     DWORD PTR [esp+0x4],eax
0x00001f7d   call    0x1f94 <dyld_stub_printf>
0x00001f82   mov     ecx,0x0
0x00001f87   mov     DWORD PTR [ebp-0x14],eax
0x00001f8a   mov     eax,ecx
0x00001f8c   add     esp,0x20
0x00001f8f   pop     esi
0x00001f90   pop     edi
0x00001f91   pop     ebp
0x00001f92   ret
```

```c
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

int main(void)
{
    int i = a[0] + 1;
    int n = i + a[++i] + b[++i];
    printf("%d\n", n);
}
```

seems like clang is doing a typical left-to-right evaluation. Since the C standard does not impose a particular evaluation order, then clang can do whatever it wants, even giving the value that most programmers would expect.

clang

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   push   edi
0x00001f24   push   esi
0x00001f25   sub    esp,0x20
0x00001f28   call   0x1f2d <main+13>
0x00001f2d   pop    eax
0x00001f2e   lea    ecx,[eax+0x85]
0x00001f34   mov    edx,DWORD PTR [eax+0xdf]
0x00001f3a   add    edx,0x1
0x00001f40   mov    DWORD PTR [ebp-0xc],edx
0x00001f43   mov    edx,DWORD PTR [ebp-0xc]
0x00001f46   mov    esi,DWORD PTR [ebp-0xc]
0x00001f49   mov    edi,esi
0x00001f4b   add    edi,0x1
0x00001f51   mov    DWORD PTR [ebp-0xc],edi
0x00001f54   add    edx,DWORD PTR [eax+esi*4+0xe3]
0x00001f5b   mov    esi,DWORD PTR [ebp-0xc]
0x00001f5e   mov    edi,esi
0x00001f60   add    edi,0x1
0x00001f66   mov    DWORD PTR [ebp-0xc],edi
0x00001f69   add    edx,DWORD PTR [eax+esi*4+0xf7]
0x00001f70   mov    DWORD PTR [ebp-0x10],edx
0x00001f73   mov    eax,DWORD PTR [ebp-0x10]
0x00001f76   mov    DWORD PTR [esp],ecx
0x00001f79   mov    DWORD PTR [esp+0x4],eax
0x00001f7d   call   0x1f94 <dyld_stub_printf>
0x00001f82   mov    ecx,0x0
0x00001f87   mov    DWORD PTR [ebp-0x14],eax
0x00001f8a   mov    eax,ecx
0x00001f8c   add    esp,0x20
0x00001f8f   pop    esi
0x00001f90   pop    edi
0x00001f91   pop    ebp
0x00001f92   ret
```

```
#include <stdio.h>

int a[] = {0,2,4,6,8};
int b[] = {0,2,4,6,8};

struct tmp {int x,y,z;} tmp;

int main(void)
{
    int i;
    int n;

    i = a[0] + 1;

    tmp.x = i;
    tmp.y = i;
    ++i;
    tmp.x += *(a+1 + tmp.y);
    tmp.y = i;
    ++i;
    tmp.x += *(b+1 + tmp.y);

    n = tmp.x;

    printf("%d\n", n);
}
```

```
n = i + a[++i] + b[++i];
```

```
n = i + a[++i] + b[++i];
```

This is how clang interprets this expression

```
n = i + a[++i] + b[++i];
```

clang

```
tmp.x = i;
tmp.y = i;
++i;
tmp.x += *(a+1 + tmp.y);
tmp.y = i;
++i;
tmp.x += *(b+1 + tmp.y);
```

This is how clang interprets this expression

```
n = i + a[++i] + b[++i];
```

clang

```
tmp.x = i;
tmp.y = i;
++i;
tmp.x += *(a+1 + tmp.y);
tmp.y = i;
++i;
tmp.x += *(b+1 + tmp.y);
```

This is how clang interprets this expression

clang

```
i + a[++i] + b[++i]
1 + a[++i] + b[++i]
1 + a[++1] + b[++i]
1 + a[2] + b[++i]
1 + 4 + b[++i]
5 + b[++i]
5 + b[++2]
5 + b[3]
5 + 6
11
```

How three popular compilers treat an expression with sequence point violation.

```
n = i + a[++i] + b[++i];
```

gcc

```
tmp.x = a[++i];
tmp.x += i;
tmp.y = b[++i];
n = tmp.x + tmp.y;
```

clang

```
tmp.x = i;
tmp.y = i;
++i;
tmp.x += *(a + 1 + tmp.y);
tmp.y = i;
++i;
tmp.x += *(b + 1 + tmp.y);
```

icc

```
i = a[0] + 1;
tmp.x = ++i;
tmp.y = ++i;
tmp.z = a[tmp.x];
tmp.z += i;
tmp.z += b[tmp.y];
n = tmp.z;
```

```
i + a[++i] + b[++i]
i + a[++1] + b[++i]
i + a[2] + b[++i]
i + 4 + b[++i]
2 + 4 + b[++i]
6 + b[++i]
6 + b[++2]
6 + b[3]
6 + 6
12
```

```
i + a[++i] + b[++i]
1 + a[++i] + b[++i]
1 + a[++1] + b[++i]
1 + a[2] + b[++i]
1 + 4 + b[++i]
5 + b[++i]
5 + b[++2]
5 + b[3]
5 + 6
11
```

```
i + a[++i] + b[++i]
i + a[++1] + b[++i]
i + a[2] + b[++i]
i + a[2] + b[++2]
i + a[2] + b[3]
3 + 4 + b[3]
7 + b[3]
7 + 6
13
```

Exercise

This code is **undefined behavior** because b is used without being initialized (it has an indeterminate value). But in practice, what do you think are possible outcomes when this function is called?

foo.c

```
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

Exercise

This code is **undefined behavior** because b is used without being initialized (it has an indeterminate value). But in practice, what do you think are possible outcomes when this function is called?

foo.c

```
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

main.c

```
void bar(void);
void foo(void);

int main(void)
{
    bar();
    foo();
}
```

Exercise

This code is **undefined behavior** because `b` is used without being initialized (it has an indeterminate value). But in practice, what do you think are possible outcomes when this function is called?

foo.c

```
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

main.c

```
void bar(void);
void foo(void);

int main(void)
{
    bar();
    foo();
}
```

bar.c

```
void bar(void)
{
    char c = 2;
    (void)c;
}
```

Exercise

This code is **undefined behavior** because `b` is used without being initialized (it has an indeterminate value). But in practice, what do you think are possible outcomes when this function is called?

bar.c

```
void bar(void)
{
    char c = 2;
    (void)c;
}
```

foo.c

```
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

main.c

```
void bar(void);
void foo(void);

int main(void)
{
    bar();
    foo();
}
```

This is what I get on my computer with no optimization (`-O0 -m32 -mtune=i386`) :

Exercise

This code is **undefined behavior** because `b` is used without being initialized (it has an indeterminate value). But in practice, what do you think are possible outcomes when this function is called?

bar.c

```
void bar(void)
{
    char c = 2;
    (void)c;
}
```

foo.c

```
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

main.c

```
void bar(void);
void foo(void);

int main(void)
{
    bar();
    foo();
}
```

This is what I get on my computer with no optimization (`-O0 -m32 -mtune=i386`) :

icc 13.0.1

```
true
```

Exercise

This code is **undefined behavior** because `b` is used without being initialized (it has an indeterminate value). But in practice, what do you think are possible outcomes when this function is called?

foo.c

```c
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

main.c

```c
void bar(void);
void foo(void);

int main(void)
{
    bar();
    foo();
}
```

bar.c

```c
void bar(void)
{
    char c = 2;
    (void)c;
}
```

This is what I get on my computer with no optimization (`-O0 -m32 -mtune=i386`) :

icc 13.0.1

```
true
```

clang 4.1

```
false
```

Exercise

This code is **undefined behavior** because `b` is used without being initialized (it has an indeterminate value). But in practice, what do you think are possible outcomes when this function is called?

foo.c

```
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

main.c

```
void bar(void);
void foo(void);

int main(void)
{
    bar();
    foo();
}
```

bar.c

```
void bar(void)
{
    char c = 2;
    (void)c;
}
```

This is what I get on my computer with no optimization (`-O0 -m32 -mtune=i386`) :

icc 13.0.1

```
true
```

clang 4.1

```
false
```

gcc 4.7.2

```
true
false
```

Exercise

This code is **undefined behavior** because b is used without being initialized (it has an indeterminate value). But in practice, what do you think are possible outcomes when this function is called?

bar.c

```
void bar(void)
{
    char c = 2;
    (void)c;
}
```

foo.c

```
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

main.c

```
void bar(void);
void foo(void);

int main(void)
{
    bar();
    foo();
}
```

This is what I get on my computer with no optimization (`-O0 -m32 -mtune=i386`) :

icc 13.0.1

```
true
```

clang 4.1

```
false
```

gcc 4.7.2

```
true
false
```

(thanks to Mark Shroyer for blogging about this very interesting gcc behavior http://markshroyer.com/2012/06/c-both-true-and-false/)

Exercise

This code is **undefined behavior** because `b` is used without being initialized (it has an indeterminate value). But in practice, what do you think are possible outcomes when this function is called?

bar.c

```
void bar(void)
{
    char c = 2;
    (void)c;
}
```

foo.c

```
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

main.c

```
void bar(void);
void foo(void);

int main(void)
{
    bar();
    foo();
}
```

This is what I get on my computer with no optimization (`-O0 -m32 -mtune=i386`) :

icc 13.0.1

```
true
```

clang 4.1

```
false
```

gcc 4.7.2

```
true
false
```

with optimization (–O2) I get:

(thanks to Mark Shroyer for blogging about this very interesting gcc behavior http://markshroyer.com/2012/06/c-both-true-and-false/)

Exercise

This code is **undefined behavior** because `b` is used without being initialized (it has an indeterminate value). But in practice, what do you think are possible outcomes when this function is called?

bar.c

```
void bar(void)
{
    char c = 2;
    (void)c;
}
```

foo.c

```
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

main.c

```
void bar(void);
void foo(void);

int main(void)
{
    bar();
    foo();
}
```

This is what I get on my computer with no optimization (`–O0 –m32 –mtune=i386`) :

icc 13.0.1

```
true
```

clang 4.1

```
false
```

gcc 4.7.2

```
true
false
```

with optimization (–O2) I get:

```
false
```

(thanks to Mark Shroyer for blogging about this very interesting gcc behavior http://markshroyer.com/2012/06/c-both-true-and-false/)

Exercise

This code is **undefined behavior** because `b` is used without being initialized (it has an indeterminate value). But in practice, what do you think are possible outcomes when this function is called?

bar.c

```
void bar(void)
{
    char c = 2;
    (void)c;
}
```

foo.c

```
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

main.c

```
void bar(void);
void foo(void);

int main(void)
{
    bar();
    foo();
}
```

This is what I get on my computer with no optimization (`-O0 -m32 -mtune=i386`) :

icc 13.0.1

```
true
```

clang 4.1

```
false
```

gcc 4.7.2

```
true
false
```

with optimization (`-O2`) I get:

```
false
```

```
false
```

(thanks to Mark Shroyer for blogging about this very interesting gcc behavior http://markshroyer.com/2012/06/c-both-true-and-false/)

Exercise

This code is **undefined behavior** because `b` is used without being initialized (it has an indeterminate value). But in practice, what do you think are possible outcomes when this function is called?

bar.c

```
void bar(void)
{
    char c = 2;
    (void)c;
}
```

foo.c

```
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

main.c

```
void bar(void);
void foo(void);

int main(void)
{
    bar();
    foo();
}
```

This is what I get on my computer with no optimization (`-O0 -m32 -mtune=i386`) :

icc 13.0.1

```
true
```

clang 4.1

```
false
```

gcc 4.7.2

```
true
false
```

with optimization (`-O2`) I get:

```
false
```

```
false
```

```
false
```

(thanks to Mark Shroyer for blogging about this very interesting gcc behavior http://markshroyer.com/2012/06/c-both-true-and-false/)

It is looking at assembler time!

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

```c
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

icc 13.0.1 with no optimization (–O0)

```
0x00001f5c   push    ebp
0x00001f5d   mov     ebp,esp
0x00001f5f   sub     esp,0x8
0x00001f62   call    0x1f67 <foo+11>
0x00001f67   pop     eax
0x00001f68   mov     DWORD PTR [ebp-0x4],eax
0x00001f6b   movzx   eax,BYTE PTR [ebp-0x8]
0x00001f6f   movzx   eax,al
0x00001f72   test    eax,eax
0x00001f74   je      0x1f8d <foo+49>
0x00001f76   add     esp,0xfffffff0
0x00001f79   mov     eax,DWORD PTR [ebp-0x4]
0x00001f7c   lea     eax,[eax+0x89]
0x00001f82   mov     DWORD PTR [esp],eax
0x00001f85   call    0x1fca <dyld_stub_printf>
0x00001f8a   add     esp,0x10
0x00001f8d   movzx   eax,BYTE PTR [ebp-0x8]
0x00001f91   movzx   eax,al
0x00001f94   test    eax,eax
0x00001f96   jne     0x1faf <foo+83>
0x00001f98   add     esp,0xfffffff0
0x00001f9b   mov     eax,DWORD PTR [ebp-0x4]
0x00001f9e   lea     eax,[eax+0x91]
0x00001fa4   mov     DWORD PTR [esp],eax
0x00001fa7   call    0x1fca <dyld_stub_printf>
0x00001fac   add     esp,0x10
0x00001faf   leave
0x00001fb0   ret
```

```c
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

icc 13.0.1 with no optimization (−O0)

```
0x00001f5c    push    ebp
0x00001f5d    mov     ebp,esp
0x00001f5f    sub     esp,0x8
0x00001f62    call    0x1f67 <foo+11>
0x00001f67    pop     eax
0x00001f68    mov     DWORD PTR [ebp-0x4],eax
0x00001f6b    movzx   eax,BYTE PTR [ebp-0x8]
0x00001f6f    movzx   eax,al
0x00001f72    test    eax,eax
0x00001f74    je      0x1f8d <foo+49>
0x00001f76    add     esp,0xfffffff0
0x00001f79    mov     eax,DWORD PTR [ebp-0x4]
0x00001f7c    lea     eax,[eax+0x89]
0x00001f82    mov     DWORD PTR [esp],eax
0x00001f85    call    0x1fca <dyld_stub_printf>
0x00001f8a    add     esp,0x10
0x00001f8d    movzx   eax,BYTE PTR [ebp-0x8]
0x00001f91    movzx   eax,al
0x00001f94    test    eax,eax
0x00001f96    jne     0x1faf <foo+83>
0x00001f98    add     esp,0xfffffff0
0x00001f9b    mov     eax,DWORD PTR [ebp-0x4]
0x00001f9e    lea     eax,[eax+0x91]
0x00001fa4    mov     DWORD PTR [esp],eax
0x00001fa7    call    0x1fca <dyld_stub_printf>
0x00001fac    add     esp,0x10
0x00001faf    leave
0x00001fb0    ret
```

icc 13.0.1 with no optimization (−O0)

```c
void foo(void) {
    char b; // “random” value
    reg.a = b;
    if (reg.a == 0)
        goto label1;
    printf("true\n");
  label1:
    reg.a = b;
    if (reg.a != 0)
        goto label2;
    printf("false\n");
  label2:
    ;
}
```

```c
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

icc 13.0.1 with no optimization (–O0)

```
0x00001f5c   push   ebp
0x00001f5d   mov    ebp,esp
0x00001f5f   sub    esp,0x8
0x00001f62   call   0x1f67 <foo+11>
0x00001f67   pop    eax
0x00001f68   mov    DWORD PTR [ebp-0x4],eax
0x00001f6b   movzx  eax,BYTE PTR [ebp-0x8]
0x00001f6f   movzx  eax,al
0x00001f72   test   eax,eax
0x00001f74   je     0x1f8d <foo+49>
0x00001f76   add    esp,0xfffffff0
0x00001f79   mov    eax,DWORD PTR [ebp-0x4]
0x00001f7c   lea    eax,[eax+0x89]
0x00001f82   mov    DWORD PTR [esp],eax
0x00001f85   call   0x1fca <dyld_stub_printf>
0x00001f8a   add    esp,0x10
0x00001f8d   movzx  eax,BYTE PTR [ebp-0x8]
0x00001f91   movzx  eax,al
0x00001f94   test   eax,eax
0x00001f96   jne    0x1faf <foo+83>
0x00001f98   add    esp,0xfffffff0
0x00001f9b   mov    eax,DWORD PTR [ebp-0x4]
0x00001f9e   lea    eax,[eax+0x91]
0x00001fa4   mov    DWORD PTR [esp],eax
0x00001fa7   call   0x1fca <dyld_stub_printf>
0x00001fac   add    esp,0x10
0x00001faf   leave
0x00001fb0   ret
```

icc 13.0.1 with no optimization (–O0)

```c
void foo(void) {
    char b; // "random" value
    reg.a = b;
    if (reg.a == 0)
        goto label1;
    printf("true\n");
  label1:
    reg.a = b;
    if (reg.a != 0)
        goto label2;
    printf("false\n");
  label2:
    ;
}
```

icc is doing what most programmers would expect might happen

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

icc 13.0.1 with no optimization (–O0)

```
0x00001f5c   push    ebp
0x00001f5d   mov     ebp,esp
0x00001f5f   sub     esp,0x8
0x00001f62   call    0x1f67 <foo+11>
0x00001f67   pop     eax
0x00001f68   mov     DWORD PTR [ebp-0x4],eax
0x00001f6b   movzx   eax,BYTE PTR [ebp-0x8]
0x00001f6f   movzx   eax,al
0x00001f72   test    eax,eax
0x00001f74   je      0x1f8d <foo+49>
0x00001f76   add     esp,0xfffffff0
0x00001f79   mov     eax,DWORD PTR [ebp-0x4]
0x00001f7c   lea     eax,[eax+0x89]
0x00001f82   mov     DWORD PTR [esp],eax
0x00001f85   call    0x1fca <dyld_stub_printf>
0x00001f8a   add     esp,0x10
0x00001f8d   movzx   eax,BYTE PTR [ebp-0x8]
0x00001f91   movzx   eax,al
0x00001f94   test    eax,eax
0x00001f96   jne     0x1faf <foo+83>
0x00001f98   add     esp,0xfffffff0
0x00001f9b   mov     eax,DWORD PTR [ebp-0x4]
0x00001f9e   lea     eax,[eax+0x91]
0x00001fa4   mov     DWORD PTR [esp],eax
0x00001fa7   call    0x1fca <dyld_stub_printf>
0x00001fac   add     esp,0x10
0x00001faf   leave
0x00001fb0   ret
```

icc 13.0.1 with no optimization (–O0)

```
void foo(void) {
    char b; // "random" value
    reg.a = b;
    if (reg.a == 0)
        goto label1;
    printf("true\n");
  label1:
    reg.a = b;
    if (reg.a != 0)
        goto label2;
    printf("false\n");
  label2:
    ;
}
```

"Random" value	output
0	false
1	true
anything else	true

icc is doing what most programmers would expect might happen

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

```c
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

icc 13.0.1 with optimization (–02)

```
0x00001e40   sub    esp,0x1c
0x00001e43   call   0x1e48 <foo+8>
0x00001e48   pop    edx
0x00001e49   test   al,al
0x00001e4b   jne    0x1e60 <foo+32>
0x00001e4d   add    esp,0x4
0x00001e50   lea    eax,[edx+0x1ac]
0x00001e56   push   eax
0x00001e57   call   0x1fc6 <dyld_stub_printf>
0x00001e5c   add    esp,0x1c
0x00001e5f   ret
0x00001e60   add    esp,0x4
0x00001e63   lea    eax,[edx+0x1a4]
0x00001e69   push   eax
0x00001e6a   call   0x1fc6 <dyld_stub_printf>
0x00001e6f   add    esp,0x1c
0x00001e72   ret
```

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

icc 13.0.1 with optimization (–02)

```
0x00001e40    sub     esp,0x1c
0x00001e43    call    0x1e48 <foo+8>
0x00001e48    pop     edx
0x00001e49    test    al,al
0x00001e4b    jne     0x1e60 <foo+32>
0x00001e4d    add     esp,0x4
0x00001e50    lea     eax,[edx+0x1ac]
0x00001e56    push    eax
0x00001e57    call    0x1fc6 <dyld_stub_printf>
0x00001e5c    add     esp,0x1c
0x00001e5f    ret
0x00001e60    add     esp,0x4
0x00001e63    lea     eax,[edx+0x1a4]
0x00001e69    push    eax
0x00001e6a    call    0x1fc6 <dyld_stub_printf>
0x00001e6f    add     esp,0x1c
0x00001e72    ret
```

icc 13.0.1 with optimization (–02)

```
void foo(void) {
    reg.a; // “random” value
    if (reg.a != 0)
        goto label1;
    printf("false\n");
    return;
  label1:
    printf("true\n");
    return;
}
```

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

icc 13.0.1 with optimization (–02)

```
0x00001e40    sub     esp,0x1c
0x00001e43    call    0x1e48 <foo+8>
0x00001e48    pop     edx
0x00001e49    test    al,al
0x00001e4b    jne     0x1e60 <foo+32>
0x00001e4d    add     esp,0x4
0x00001e50    lea     eax,[edx+0x1ac]
0x00001e56    push    eax
0x00001e57    call    0x1fc6 <dyld_stub_printf>
0x00001e5c    add     esp,0x1c
0x00001e5f    ret
0x00001e60    add     esp,0x4
0x00001e63    lea     eax,[edx+0x1a4]
0x00001e69    push    eax
0x00001e6a    call    0x1fc6 <dyld_stub_printf>
0x00001e6f    add     esp,0x1c
0x00001e72    ret
```

icc 13.0.1 with optimization (–02)

```
void foo(void) {
    reg.a; // “random” value
    if (reg.a != 0)
        goto label1;
    printf("false\n");
    return;
  label1:
    printf("true\n");
    return;
}
```

Notice that icc does not even create space for the variable b. It is just using the random value stored in the eax register.

```c
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

icc 13.0.1 with optimization (–02)

```
0x00001e40    sub     esp,0x1c
0x00001e43    call    0x1e48 <foo+8>
0x00001e48    pop     edx
0x00001e49    test    al,al
0x00001e4b    jne     0x1e60 <foo+32>
0x00001e4d    add     esp,0x4
0x00001e50    lea     eax,[edx+0x1ac]
0x00001e56    push    eax
0x00001e57    call    0x1fc6 <dyld_stub_printf>
0x00001e5c    add     esp,0x1c
0x00001e5f    ret
0x00001e60    add     esp,0x4
0x00001e63    lea     eax,[edx+0x1a4]
0x00001e69    push    eax
0x00001e6a    call    0x1fc6 <dyld_stub_printf>
0x00001e6f    add     esp,0x1c
0x00001e72    ret
```

icc 13.0.1 with optimization (–02)

```c
void foo(void) {
    reg.a; // “random” value
    if (reg.a != 0)
        goto label1;
    printf("false\n");
    return;
  label1:
    printf("true\n");
    return;
}
```

“Random” value	**output**
0	false
1	true
anything else	true

Notice that icc does not even create space for the variable b. It is just using the random value stored in the eax register.

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

clang 4.1 with no optimization (–O0)

```
0x00001f20    push    ebp
0x00001f21    mov     ebp,esp
0x00001f23    sub     esp,0x18
0x00001f26    call    0x1f2b <foo+11>
0x00001f2b    pop     eax
0x00001f2c    test    BYTE PTR [ebp-0x1],0x1
0x00001f30    mov     DWORD PTR [ebp-0x8],eax
0x00001f33    je      0x1f4d <foo+45>
0x00001f39    mov     eax,DWORD PTR [ebp-0x8]
0x00001f3c    lea     ecx,[eax+0x73]
0x00001f42    mov     DWORD PTR [esp],ecx
0x00001f45    call    0x1f80 <dyld_stub_printf>
0x00001f4a    mov     DWORD PTR [ebp-0xc],eax
0x00001f4d    test    BYTE PTR [ebp-0x1],0x1
0x00001f51    jne     0x1f6b <foo+75>
0x00001f57    mov     eax,DWORD PTR [ebp-0x8]
0x00001f5a    lea     ecx,[eax+0x79]
0x00001f60    mov     DWORD PTR [esp],ecx
0x00001f63    call    0x1f80 <dyld_stub_printf>
0x00001f68    mov     DWORD PTR [ebp-0x10],eax
0x00001f6b    add     esp,0x18
0x00001f6e    pop     ebp
0x00001f6f    ret
```

```c
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

clang 4.1 with no optimization (–O0)

```
0x00001f20    push    ebp
0x00001f21    mov     ebp,esp
0x00001f23    sub     esp,0x18
0x00001f26    call    0x1f2b <foo+11>
0x00001f2b    pop     eax
0x00001f2c    test    BYTE PTR [ebp-0x1],0x1
0x00001f30    mov     DWORD PTR [ebp-0x8],eax
0x00001f33    je      0x1f4d <foo+45>
0x00001f39    mov     eax,DWORD PTR [ebp-0x8]
0x00001f3c    lea     ecx,[eax+0x73]
0x00001f42    mov     DWORD PTR [esp],ecx
0x00001f45    call    0x1f80 <dyld_stub_printf>
0x00001f4a    mov     DWORD PTR [ebp-0xc],eax
0x00001f4d    test    BYTE PTR [ebp-0x1],0x1
0x00001f51    jne     0x1f6b <foo+75>
0x00001f57    mov     eax,DWORD PTR [ebp-0x8]
0x00001f5a    lea     ecx,[eax+0x79]
0x00001f60    mov     DWORD PTR [esp],ecx
0x00001f63    call    0x1f80 <dyld_stub_printf>
0x00001f68    mov     DWORD PTR [ebp-0x10],eax
0x00001f6b    add     esp,0x18
0x00001f6e    pop     ebp
0x00001f6f    ret
```

clang 4.1 with no optimization (–O0)

```c
void foo(void) {
    char b; // “random” value
    if ((b & 1) != 1)
        goto label1;
    printf("true\n");
  label1:
    if ((b & 1) == 1)
        goto label2;
    printf("false\n");
  label2:
    ;
}
```

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

clang 4.1 with no optimization (–O0)

```
0x00001f20    push    ebp
0x00001f21    mov     ebp,esp
0x00001f23    sub     esp,0x18
0x00001f26    call    0x1f2b <foo+11>
0x00001f2b    pop     eax
0x00001f2c    test    BYTE PTR [ebp-0x1],0x1
0x00001f30    mov     DWORD PTR [ebp-0x8],eax
0x00001f33    je      0x1f4d <foo+45>
0x00001f39    mov     eax,DWORD PTR [ebp-0x8]
0x00001f3c    lea     ecx,[eax+0x73]
0x00001f42    mov     DWORD PTR [esp],ecx
0x00001f45    call    0x1f80 <dyld_stub_printf>
0x00001f4a    mov     DWORD PTR [ebp-0xc],eax
0x00001f4d    test    BYTE PTR [ebp-0x1],0x1
0x00001f51    jne     0x1f6b <foo+75>
0x00001f57    mov     eax,DWORD PTR [ebp-0x8]
0x00001f5a    lea     ecx,[eax+0x79]
0x00001f60    mov     DWORD PTR [esp],ecx
0x00001f63    call    0x1f80 <dyld_stub_printf>
0x00001f68    mov     DWORD PTR [ebp-0x10],eax
0x00001f6b    add     esp,0x18
0x00001f6e    pop     ebp
0x00001f6f    ret
```

clang 4.1 with no optimization (–O0)

```
void foo(void) {
    char b; // “random” value
    if ((b & 1) != 1)
        goto label1;
    printf("true\n");
  label1:
    if ((b & 1) == 1)
        goto label2;
    printf("false\n");
  label2:
    ;
}
```

clang just tests the last bit of the byte it uses to represent the bool.

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

clang 4.1 with no optimization (–O0)

```
0x00001f20   push   ebp
0x00001f21   mov    ebp,esp
0x00001f23   sub    esp,0x18
0x00001f26   call   0x1f2b <foo+11>
0x00001f2b   pop    eax
0x00001f2c   test   BYTE PTR [ebp-0x1],0x1
0x00001f30   mov    DWORD PTR [ebp-0x8],eax
0x00001f33   je     0x1f4d <foo+45>
0x00001f39   mov    eax,DWORD PTR [ebp-0x8]
0x00001f3c   lea    ecx,[eax+0x73]
0x00001f42   mov    DWORD PTR [esp],ecx
0x00001f45   call   0x1f80 <dyld_stub_printf>
0x00001f4a   mov    DWORD PTR [ebp-0xc],eax
0x00001f4d   test   BYTE PTR [ebp-0x1],0x1
0x00001f51   jne    0x1f6b <foo+75>
0x00001f57   mov    eax,DWORD PTR [ebp-0x8]
0x00001f5a   lea    ecx,[eax+0x79]
0x00001f60   mov    DWORD PTR [esp],ecx
0x00001f63   call   0x1f80 <dyld_stub_printf>
0x00001f68   mov    DWORD PTR [ebp-0x10],eax
0x00001f6b   add    esp,0x18
0x00001f6e   pop    ebp
0x00001f6f   ret
```

clang 4.1 with no optimization (–O0)

```
void foo(void) {
    char b; // “random” value
    if ((b & 1) != 1)
        goto label1;
    printf("true\n");
  label1:
    if ((b & 1) == 1)
        goto label2;
    printf("false\n");
  label2:
    ;
}
```

“Random” value	output
even number	false
odd number	true

clang just tests the last bit of the byte it uses to represent the bool.

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

```c
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

clang 4.1 with optimization (–02)

```
0x00001f70   push   ebp
0x00001f71   mov    ebp,esp
0x00001f73   sub    esp,0x8
0x00001f76   call   0x1f7b <foo+11>
0x00001f7b   pop    eax
0x00001f7c   lea    eax,[eax+0x37]
0x00001f82   mov    DWORD PTR [esp],eax
0x00001f85   call   0x1f96 <dyld_stub_puts>
0x00001f8a   add    esp,0x8
0x00001f8d   pop    ebp
0x00001f8e   ret
```

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

clang 4.1 with optimization (–02)

```
0x00001f70   push   ebp
0x00001f71   mov    ebp,esp
0x00001f73   sub    esp,0x8
0x00001f76   call   0x1f7b <foo+11>
0x00001f7b   pop    eax
0x00001f7c   lea    eax,[eax+0x37]
0x00001f82   mov    DWORD PTR [esp],eax
0x00001f85   call   0x1f96 <dyld_stub_puts>
0x00001f8a   add    esp,0x8
0x00001f8d   pop    ebp
0x00001f8e   ret
```

clang 4.1 with optimization (–02)

```
void foo(void) {
    puts("false");
}
```

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

clang 4.1 with optimization (–02)

```
0x00001f70   push   ebp
0x00001f71   mov    ebp,esp
0x00001f73   sub    esp,0x8
0x00001f76   call   0x1f7b <foo+11>
0x00001f7b   pop    eax
0x00001f7c   lea    eax,[eax+0x37]
0x00001f82   mov    DWORD PTR [esp],eax
0x00001f85   call   0x1f96 <dyld_stub_puts>
0x00001f8a   add    esp,0x8
0x00001f8d   pop    ebp
0x00001f8e   ret
```

clang 4.1 with optimization (–02)

```
void foo(void) {
    puts("false");
}
```

clang just prints “false”. Simple and clean!

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

clang 4.1 with optimization (–02)

```
0x00001f70   push   ebp
0x00001f71   mov    ebp,esp
0x00001f73   sub    esp,0x8
0x00001f76   call   0x1f7b <foo+11>
0x00001f7b   pop    eax
0x00001f7c   lea    eax,[eax+0x37]
0x00001f82   mov    DWORD PTR [esp],eax
0x00001f85   call   0x1f96 <dyld_stub_puts>
0x00001f8a   add    esp,0x8
0x00001f8d   pop    ebp
0x00001f8e   ret
```

clang 4.1 with optimization (–02)

```
void foo(void) {
    puts("false");
}
```

clang just prints “false”. Simple and clean!

“Random” value	**output**
any number	false

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

```c
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

gcc 4.7.2 with no optimization (–O0)

```
0x00001e98   push   ebp
0x00001e99   mov    ebp,esp
0x00001e9b   push   ebx
0x00001e9c   sub    esp,0x24
0x00001e9f   call   0x1ed6 <__x86.get_pc_thunk.bx>
0x00001ea4   cmp    BYTE PTR [ebp-0x9],0x0
0x00001ea8   je     0x1eb8 <foo+32>
0x00001eaa   lea    eax,[ebx+0x5e]
0x00001eb0   mov    DWORD PTR [esp],eax
0x00001eb3   call   0x1ee6 <dyld_stub_puts>
0x00001eb8   mov    al,BYTE PTR [ebp-0x9]
0x00001ebb   xor    eax,0x1
0x00001ebe   test   al,al
0x00001ec0   je     0x1ed0 <foo+56>
0x00001ec2   lea    eax,[ebx+0x63]
0x00001ec8   mov    DWORD PTR [esp],eax
0x00001ecb   call   0x1ee6 <dyld_stub_puts>
0x00001ed0   add    esp,0x24
0x00001ed3   pop    ebx
0x00001ed4   pop    ebp
0x00001ed5   ret
```

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

gcc 4.7.2 with no optimization (–O0)

```
0x00001e98   push   ebp
0x00001e99   mov    ebp,esp
0x00001e9b   push   ebx
0x00001e9c   sub    esp,0x24
0x00001e9f   call   0x1ed6 <__x86.get_pc_thunk.bx>
0x00001ea4   cmp    BYTE PTR [ebp-0x9],0x0
0x00001ea8   je     0x1eb8 <foo+32>
0x00001eaa   lea    eax,[ebx+0x5e]
0x00001eb0   mov    DWORD PTR [esp],eax
0x00001eb3   call   0x1ee6 <dyld_stub_puts>
0x00001eb8   mov    al,BYTE PTR [ebp-0x9]
0x00001ebb   xor    eax,0x1
0x00001ebe   test   al,al
0x00001ec0   je     0x1ed0 <foo+56>
0x00001ec2   lea    eax,[ebx+0x63]
0x00001ec8   mov    DWORD PTR [esp],eax
0x00001ecb   call   0x1ee6 <dyld_stub_puts>
0x00001ed0   add    esp,0x24
0x00001ed3   pop    ebx
0x00001ed4   pop    ebp
0x00001ed5   ret
```

gcc 4.7.2 with no optimization (–O0)

```
void foo(void) {
    char b; // “random” value
    if (b == 0)
        goto label1;
    puts("true");
  label1:
    reg.a = b;
    reg.a ^= 1;
    if (reg.a == 0)
        goto label2;
    puts("false");
  label2:
    ;
}
```

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

gcc 4.7.2 with no optimization (–O0)

```
0x00001e98   push   ebp
0x00001e99   mov    ebp,esp
0x00001e9b   push   ebx
0x00001e9c   sub    esp,0x24
0x00001e9f   call   0x1ed6 <__x86.get_pc_thunk.bx>
0x00001ea4   cmp    BYTE PTR [ebp-0x9],0x0
0x00001ea8   je     0x1eb8 <foo+32>
0x00001eaa   lea    eax,[ebx+0x5e]
0x00001eb0   mov    DWORD PTR [esp],eax
0x00001eb3   call   0x1ee6 <dyld_stub_puts>
0x00001eb8   mov    al,BYTE PTR [ebp-0x9]
0x00001ebb   xor    eax,0x1
0x00001ebe   test   al,al
0x00001ec0   je     0x1ed0 <foo+56>
0x00001ec2   lea    eax,[ebx+0x63]
0x00001ec8   mov    DWORD PTR [esp],eax
0x00001ecb   call   0x1ee6 <dyld_stub_puts>
0x00001ed0   add    esp,0x24
0x00001ed3   pop    ebx
0x00001ed4   pop    ebp
0x00001ed5   ret
```

gcc 4.7.2 with no optimization (–O0)

```
void foo(void) {
    char b; // “random” value
    if (b == 0)
        goto label1;
    puts("true");
  label1:
    reg.a = b;
    reg.a ^= 1;
    if (reg.a == 0)
        goto label2;
    puts("false");
  label2:
    ;
}
```

gcc assumes that the bitpattern in the byte representing a bool is always 0 or 1, never anything else.

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

gcc 4.7.2 with no optimization (–O0)

```
0x00001e98   push   ebp
0x00001e99   mov    ebp,esp
0x00001e9b   push   ebx
0x00001e9c   sub    esp,0x24
0x00001e9f   call   0x1ed6 <__x86.get_pc_thunk.bx>
0x00001ea4   cmp    BYTE PTR [ebp-0x9],0x0
0x00001ea8   je     0x1eb8 <foo+32>
0x00001eaa   lea    eax,[ebx+0x5e]
0x00001eb0   mov    DWORD PTR [esp],eax
0x00001eb3   call   0x1ee6 <dyld_stub_puts>
0x00001eb8   mov    al,BYTE PTR [ebp-0x9]
0x00001ebb   xor    eax,0x1
0x00001ebe   test   al,al
0x00001ec0   je     0x1ed0 <foo+56>
0x00001ec2   lea    eax,[ebx+0x63]
0x00001ec8   mov    DWORD PTR [esp],eax
0x00001ecb   call   0x1ee6 <dyld_stub_puts>
0x00001ed0   add    esp,0x24
0x00001ed3   pop    ebx
0x00001ed4   pop    ebp
0x00001ed5   ret
```

gcc 4.7.2 with no optimization (–O0)

```
void foo(void) {
    char b; // "random" value
    if (b == 0)
        goto label1;
    puts("true");
  label1:
    reg.a = b;
    reg.a ^= 1;
    if (reg.a == 0)
        goto label2;
    puts("false");
  label2:
    ;
}
```

"Random" value	**output**
0	false
1	true
anything else	true false

gcc assumes that the bitpattern in the byte representing a bool is always 0 or 1, never anything else.

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

gcc 4.7.2 with no optimization (–O0)

```
0x00001e98   push   ebp
0x00001e99   mov    ebp,esp
0x00001e9b   push   ebx
0x00001e9c   sub    esp,0x24
0x00001e9f   call   0x1ed6 <__x86.get_pc_thunk.bx>
0x00001ea4   cmp    BYTE PTR [ebp-0x9],0x0
0x00001ea8   je     0x1eb8 <foo+32>
0x00001eaa   lea    eax,[ebx+0x5e]
0x00001eb0   mov    DWORD PTR [esp],eax
0x00001eb3   call   0x1ee6 <dyld_stub_puts>
0x00001eb8   mov    al,BYTE PTR [ebp-0x9]
0x00001ebb   xor    eax,0x1
0x00001ebe   test   al,al
0x00001ec0   je     0x1ed0 <foo+56>
0x00001ec2   lea    eax,[ebx+0x63]
0x00001ec8   mov    DWORD PTR [esp],eax
0x00001ecb   call   0x1ee6 <dyld_stub_puts>
0x00001ed0   add    esp,0x24
0x00001ed3   pop    ebx
0x00001ed4   pop    ebp
0x00001ed5   ret
```

gcc 4.7.2 with no optimization (–O0)

```
void foo(void) {
    char b; // “random” value
    if (b == 0)
        goto label1;
    puts("true");
  label1:
    reg.a = b;
    reg.a ^= 1;
    if (reg.a == 0)
        goto label2;
    puts("false");
  label2:
    ;
}
```

“Random” value	**output**
0	false
1	true
anything else	true false

gcc assumes that the bitpattern in the byte representing a bool is always 0 or 1, never anything else.

... and there is nothing wrong with that. We have broken the rules of the language by reading an unintialized object.

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

gcc 4.7.2 with optimization (–O2)

```
0x00001edc   push   ebx
0x00001edd   sub    esp,0x18
0x00001ee0   call   0x1ef8 <__x86.get_pc_thunk.bx>
0x00001ee5   lea    eax,[ebx+0x52]
0x00001eeb   mov    DWORD PTR [esp],eax
0x00001eee   call   0x1f14 <dyld_stub_puts>
0x00001ef3   add    esp,0x18
0x00001ef6   pop    ebx
0x00001ef7   ret
```

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

gcc 4.7.2 with optimization (–02)

```
0x00001edc   push   ebx
0x00001edd   sub    esp,0x18
0x00001ee0   call   0x1ef8 <__x86.get_pc_thunk.bx>
0x00001ee5   lea    eax,[ebx+0x52]
0x00001eeb   mov    DWORD PTR [esp],eax
0x00001eee   call   0x1f14 <dyld_stub_puts>
0x00001ef3   add    esp,0x18
0x00001ef6   pop    ebx
0x00001ef7   ret
```

gcc 4.7.2 with optimization (–02)

```
void foo(void) {
    puts("false");
}
```

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

gcc 4.7.2 with optimization (–02)

```
0x00001edc    push    ebx
0x00001edd    sub     esp,0x18
0x00001ee0    call    0x1ef8 <__x86.get_pc_thunk.bx>
0x00001ee5    lea     eax,[ebx+0x52]
0x00001eeb    mov     DWORD PTR [esp],eax
0x00001eee    call    0x1f14 <dyld_stub_puts>
0x00001ef3    add     esp,0x18
0x00001ef6    pop     ebx
0x00001ef7    ret
```

gcc 4.7.2 with optimization (–02)

```
void foo(void) {
    puts("false");
}
```

gcc just prints “false”.

```
void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

gcc 4.7.2 with optimization (–02)

```
0x00001edc   push   ebx
0x00001edd   sub    esp,0x18
0x00001ee0   call   0x1ef8 <__x86.get_pc_thunk.bx>
0x00001ee5   lea    eax,[ebx+0x52]
0x00001eeb   mov    DWORD PTR [esp],eax
0x00001eee   call   0x1f14 <dyld_stub_puts>
0x00001ef3   add    esp,0x18
0x00001ef6   pop    ebx
0x00001ef7   ret
```

gcc 4.7.2 with optimization (–02)

```
void foo(void) {
    puts("false");
}
```

gcc just prints “false”.

“Random” value	output
0	false
1	false
anything else	false

foo.c

```c
#include <stdio.h>
#include <stdbool.h>

void foo(void)
{
    bool b;
    if (b)
        printf("true\n");
    if (!b)
        printf("false\n");
}
```

"Random" value	icc -O0	icc -O2	clang -O0	clang -O2	gcc -O0	gcc -O2
0	false	false	false	false	false	false
1	true	true	true	false	true	false
anything else	true	true	true or false	false	true false	false

true if random value is odd
false if random value is even

Some serious words to wrap it up:

It is common to think that undefined behavior is not such a big deal, and that it is possible to reason about what the compiler might do when encountering code that break the rules. I hope I have illustrated that really strange things can happen, and will happen. It is not possible to generalize about what might happen.

While I don't show it in this presentation, it is also important to realize that undefined behavior is not only a local problem. The state of the runtime environment will be corrupted, but **also** the state of the compiler will be corrupted - meaning that UB might result in strange behavior in apparently unrelated parts of the codebase.

But, seriously, who is releasing code with undefined behavior?

But, seriously, who is releasing code with undefined behavior?

But, seriously, who is releasing code with undefined behavior?

But, seriously, who is releasing code with undefined behavior?



But, seriously, who is releasing code with undefined behavior?

But, seriously, who is releasing code with undefined behavior?






cité
de
l'espace
esa
cnes
esa
cnes


© AFP

But, seriously, who is releasing code with undefined behavior?

snippet from pftn.c in pcc 1.0.0.RELEASE 20110221

```
...
    /* if both are imag, store value, otherwise store 0.0 */
    if (!(li && ri)) {
        tfree(r);
        r = bcon(0);
    }
    p = buildtree(ASSIGN, l, r);
    p->n_type = p->n_type += (FIMAG-FLOAT);
....
```

But, seriously, who is releasing code with undefined behavior?

snippet from pftn.c in pcc 1.0.0.RELEASE 20110221

```
...
    /* if both are imag, store value, otherwise store 0.0 */
    if (!(li && ri)) {
        tfree(r);
        r = bcon(0);
    }
    p = buildtree(ASSIGN, l, r);
    p->n_type = p->n_type += (FIMAG-FLOAT);
....
```

But, seriously, who is releasing code with undefined behavior?

snippet from pftn.c in pcc 1.0.0.RELEASE 20110221

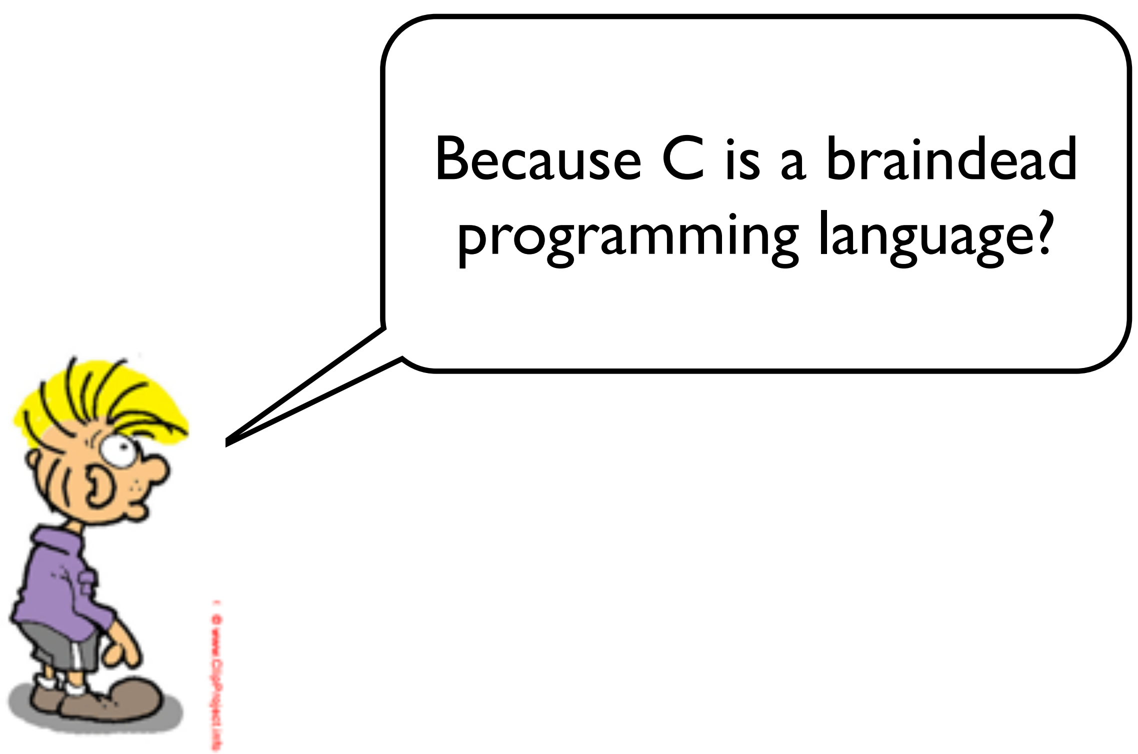
```
...
    /* if both are imag, store value, otherwise store 0.0 */
    if (!(li && ri)) {
        tfree(r);
        r = bcon(0);
    }
    p = buildtree(ASSIGN, l, r);
    p->n_type = p->n_type += (FIMAG-FLOAT);
....
```

It's undefined behavior because: "Between two sequence points, an object is modified more than once, or is modified and the prior value is read other than to determine the value to be stored
the you modify and use the value of a variable twice between sequence points"

!

In C. Why do you think static variables gets a default value (usually 0), while auto variables does not get a default value?

In C. Why do you think static variables gets a default value (usually 0), while auto variables does not get a default value?



In C. Why do you think static variables gets a default value (usually 0), while auto variables does not get a default value?

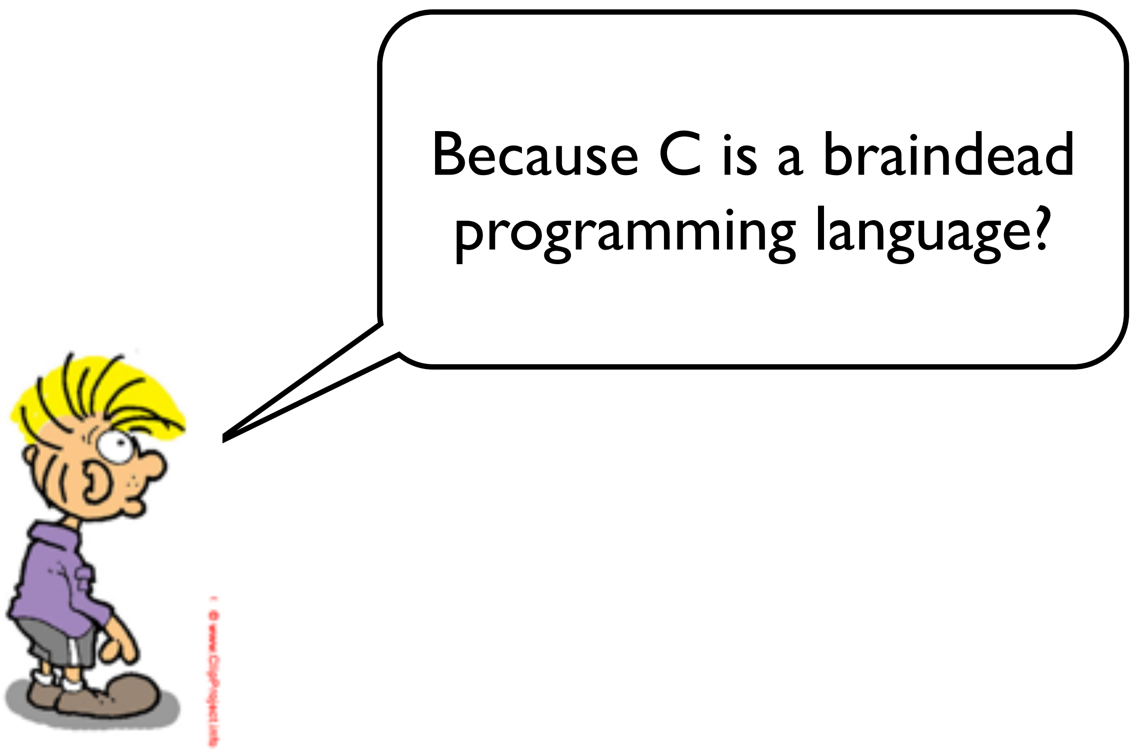
Because C is a braindead programming language?

Because C is all about execution speed. Setting static variables to default values is a one time cost, while defaulting auto variables might add a signficant runtime cost.

In C. Why is the evaluation order mostly unspecified?

In C. Why is the evaluation order mostly unspecified?

In C. Why is the evaluation order mostly unspecified?



In C. Why is the evaluation order mostly unspecified?

Because C is a braindead programming language?

Because there is a design goal to allow optimal execution speed on a wide range of architectures. In C the compiler can choose to evaluate expressions in the order that is most optimal for a particular platform. This allows for great optimization opportunities.

Why don’t the C standard require that you always get a warning or error on invalid code?

Why don't the C standard require that you always get a warning or error on invalid code?

Why don’t the C standard require that you always get a warning or error on invalid code?

Because C is a braindead programming language?

One of the design goals of C is that it should be relatively easy to write a compiler. Adding a requirement that the compilers should refuse or warn about invalid code would add a huge burden on the compiler writers.

The spirit of C

trust the programmer

- let them do what needs to be done
- the programmer is in charge not the compiler

keep the language small and simple

- small amount of code → small amount of assembler
- provide only one way to do an operation
- new inventions are not entertained

make it fast, even if its not portable

- target efficient code generation
- int preference, int promotion rules
- sequence points, maximum leeway to compiler

rich expression support

- lots of operators
- expressions combine into larger expressions