Basic C++

1

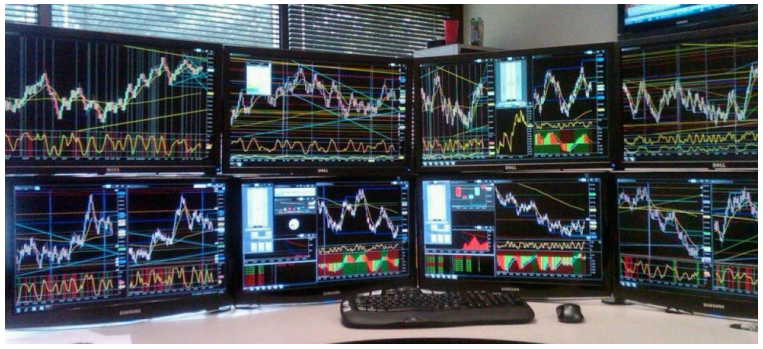
Dr. Porkoláb Zoltán Károly

gsd@inf.elte.hu

http://gsd.web.elte.hu

C++

- C++ is used in various environment
  - High performance computing
  - GPGPU programming
  - Large system implementation (like telecom)
  - Device drivers (e.g. Arduino)
  - Low energy environments (e.g. Mars rover)
  - Performance critical systems (e.g. F35)



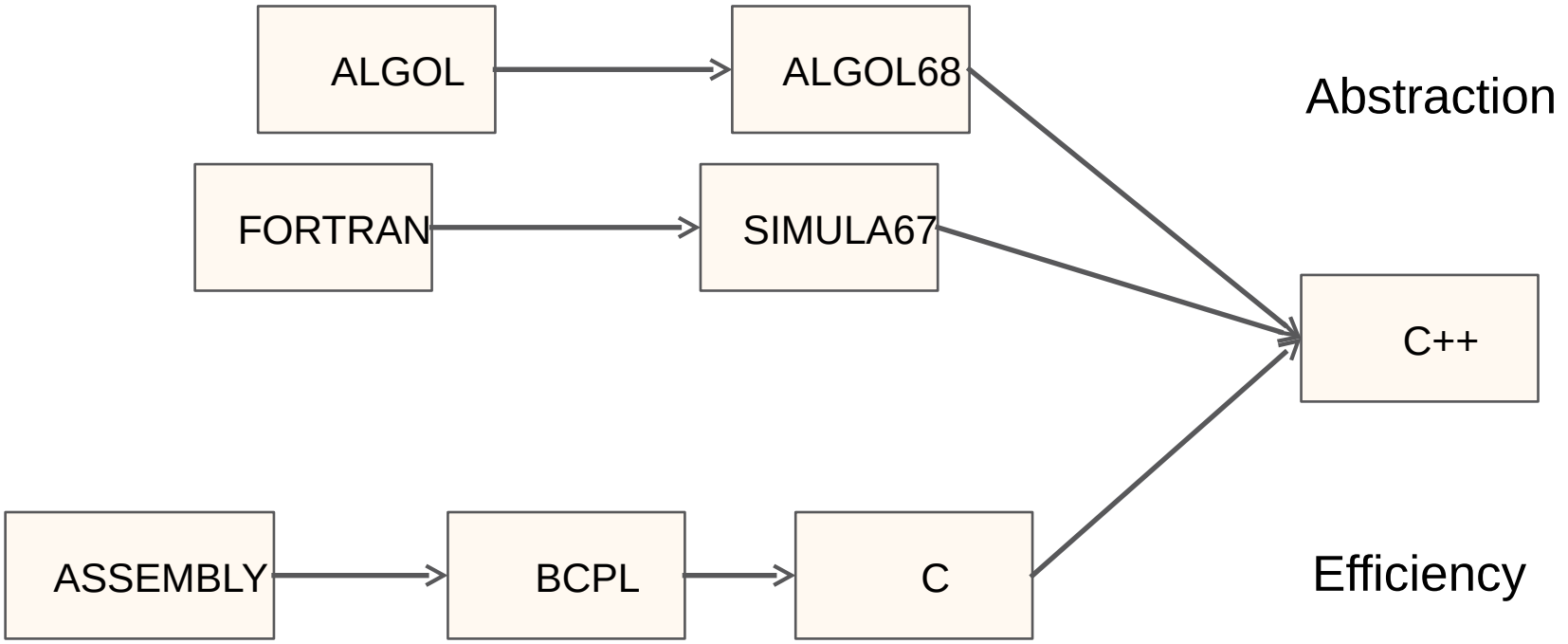
Design goals (by Stroustrup)

Type safety

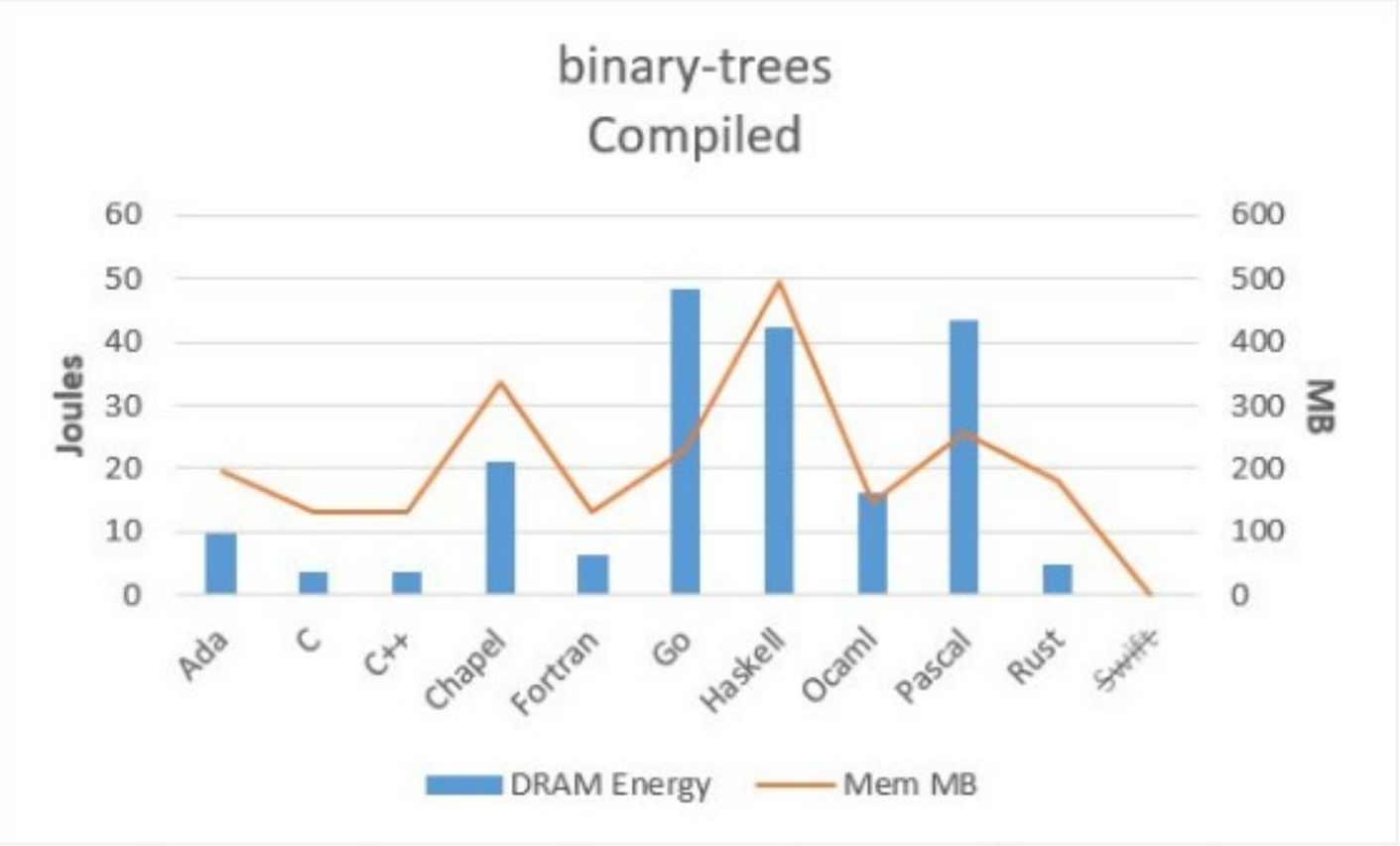
- Strong type system

- Resource safety
  - Resource > Memory
- Performance
  - Freedom for the compiler to generate the most efficient code
- Predictable run-time behavior
  - No virtual machine
  - No garbage collector
- Learnability

Origin of C++


ALGOL
ALGOL68
Abstraction
FORTRAN
SIMULA67
C++
ASSEMBLY
BCPL
C
Efficiency

Energy/Runtime/Memory efficiency



Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate?. In Proceedings of 2017 ACM SIGPLAN International Conference on Software Language Engineering (SLE'17). ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3136014.3136031

Energy/Runtime/Memory efficiency

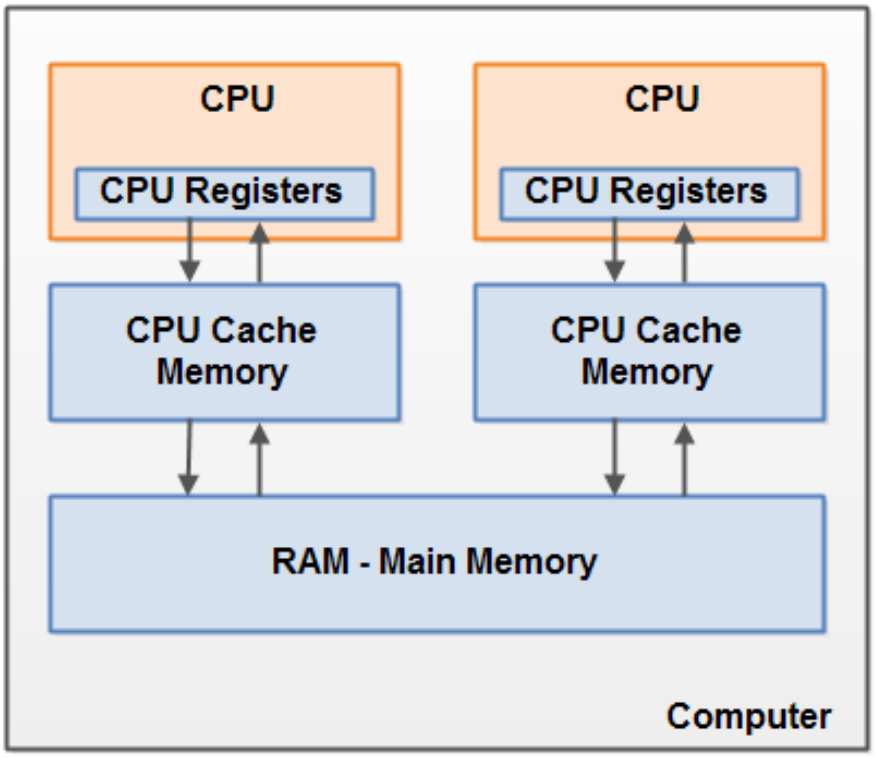
Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

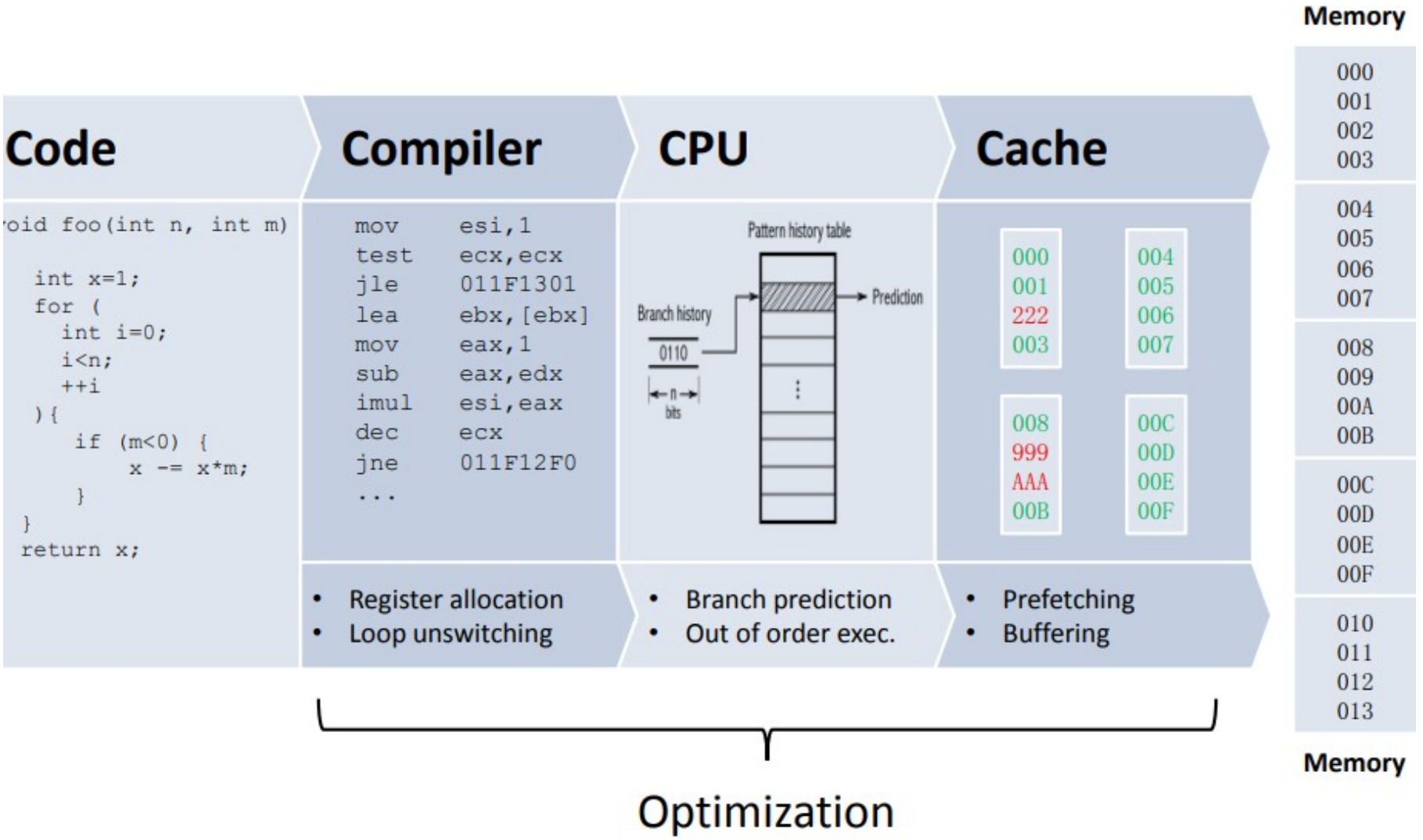
Why C++ is effective?



Action	Time
L1 cache reference	0.5ns
Branch mispredict	5ns
L2 cache reference	7ns
Mutex lock/unlock	25ns
Main memory reference	100ns
Compress 1K with Zippy	3000ns
Transfer 1K over 1Gnps net	10,000ns
Read 1Mb seq from mem.	250,000ns
Read 1Mb seq from SSD	1,000,000ns
Disk seek / Context switch	10,000,000ns
Read 1Mb seq from disk	20,000,000ns
Send packet CA->NL->CA.	150,000,000ns

https://gist.github.com/jboner/2841832

Why C++ is effective?



Why C++ is effective?

```
int f(int x, int y)
{
    x = 0;
    y = 1;
    return x;
}
```

```
1  f(int, int):
2          push    rbp
3          mov     rbp, rsp
4          mov     DWORD PTR [rbp-4], edi
5          mov     DWORD PTR [rbp-8], esi
6          mov     DWORD PTR [rbp-4], 0
7          mov     DWORD PTR [rbp-8], 1
8          mov     eax, DWORD PTR [rbp-4]
9          pop     rbp
10         ret
```

Why C++ is effective?

```
int f(int x, int y)
{
    x = 0;
    y = 1;
    return x;
}
```

```
1 f(int, int):
2         xor     eax, eax
3         ret
```

Why C++ is effective?

```
int f(int *xp, int *yp)
{
    *xp = 0;
    *yp = 1;
    return *xp;
}
```

Why C++ is effective?

```
int f(int *xp, int *yp)
{
    *xp = 0;
    *yp = 1;
    return *xp;
}
```

```
1 f(int*, int*):
2         mov     DWORD PTR [rdi], 0
3         mov     DWORD PTR [rsi], 1
4         mov     eax, DWORD PTR [rdi]
5         ret
```

Why C++ is effective?

```
int f(int *xp, long *yp)
{
    *xp = 0;
    *yp = 1;
    return *xp;
}
```

```
1  f(int*, long*):
2          mov     DWORD PTR [rdi], 0
3          xor     eax, eax
4          mov     QWORD PTR [rsi], 1
5          ret
```

Why C++ is effective?

```
int f()
{
  int sum = 0;
  for ( int i = 0; i < 100; ++i)
    sum += i;
  return sum;
}
```

Why C++ is effective?

```cpp
int f()
{
  int sum = 0;
  for ( int i = 0; i < 100; ++i)
    sum += i;
  return sum;
}
```

```
1 f():
2         mov     eax, 4950
3         ret
```

Why C++ is effective?

```cpp
int f(int n)
{
  int sum = 0;
  for ( int i = 0; i < n; ++i)
    sum += i;
  return sum;
}
```

```
1  f(int):
2          test    edi, edi
3          jle     .LBB0_1
4          lea     eax, [rdi - 1]
5          lea     ecx, [rdi - 2]
6          imul    rcx, rax
7          shr     rcx
8          lea     eax, [rcx + rdi]
9          add     eax, -1
10         ret
11 .LBB0_1:
12         xor     eax, eax
13         ret
```

What is the output?

```
void f()
{
    int i = 1;
    std::cout << i << ++i;
}
```

What is the output?

```
void f()
{
    int i = 1;
    std::cout << i << ++i;
}

$ ./a.out
12
```

What is the output?

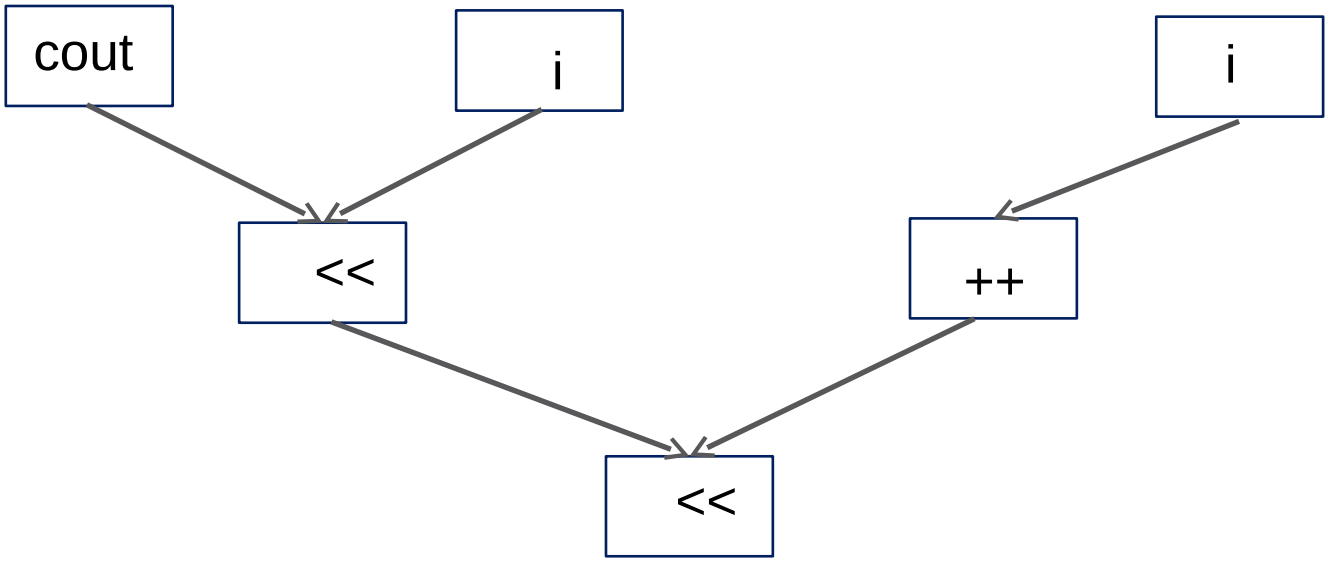
```
void f()
{
    int i = 1;
    std::cout << i << ++i;
}

$ ./a.out
22
```

What is the output?

```
void f()
{
    int i = 1;
    std::cout << i << ++i; // undefined behavior
}

$ ./a.out
12
```

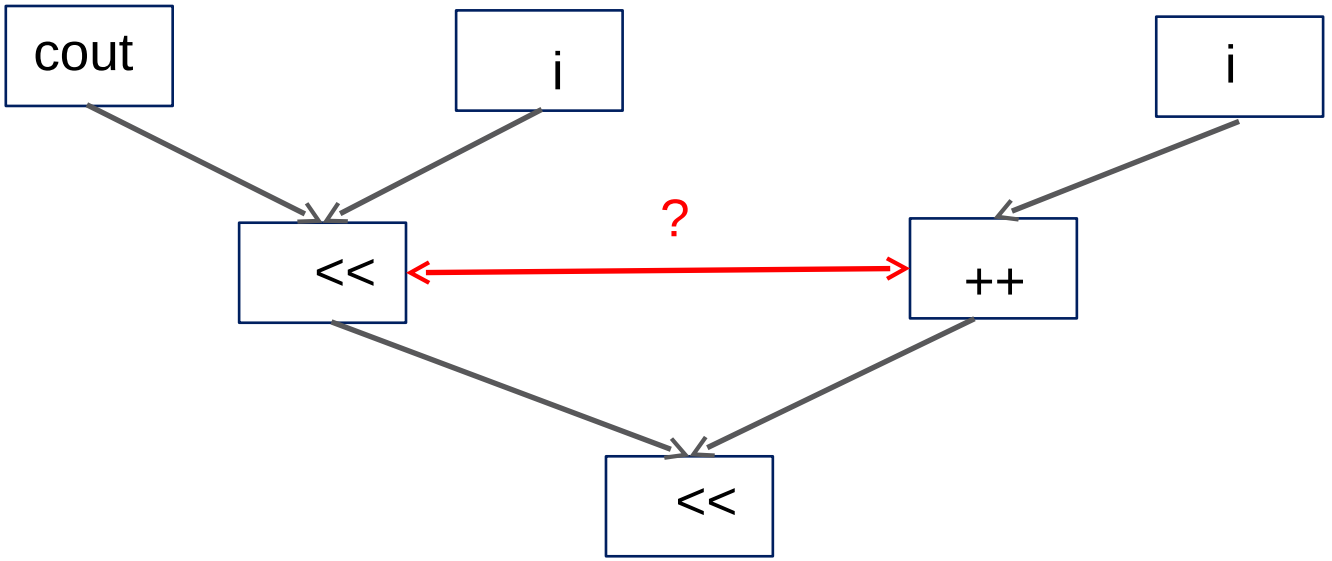
What is the output?

```
void f()
{
    int i = 1;
    std::cout << i << ++i;
}

$ ./a.out
22
```



What is the output?

```
void f()
{
    int i = 1;
    std::cout << i << ++i;
}

$ ./a.out
22
```



What is the output?

```
void f()
{
    int i = 1;
    std::cout << i << ++i;
}
```

What is the output?

```
void f()
{
    int i = 1;
    std::cout << i;
    std::cout << ++i;
}

$ ./a.out
12
```

Undefined behavior

```
int f(int x)
{
    int i;
    if ( x < 0 ) i = 1;
    return i;
}
```

Undefined behavior

```
int f(int x)
{
    int i;  // not initialized
    if ( x < 0 ) i = 1;
    return i;
}
```

```
1 f(int):
2         mov     eax, 1
3         ret
```

Hello

```
#include <iostream>

int main()
{
    std::cout << "Hello world" << std::endl;
    return 0;
}
```

Hello

```
#include <iostream>

int main()
{
    std::cout << "Hello world" << '\n'; // std::endl
    return 0;
}
```

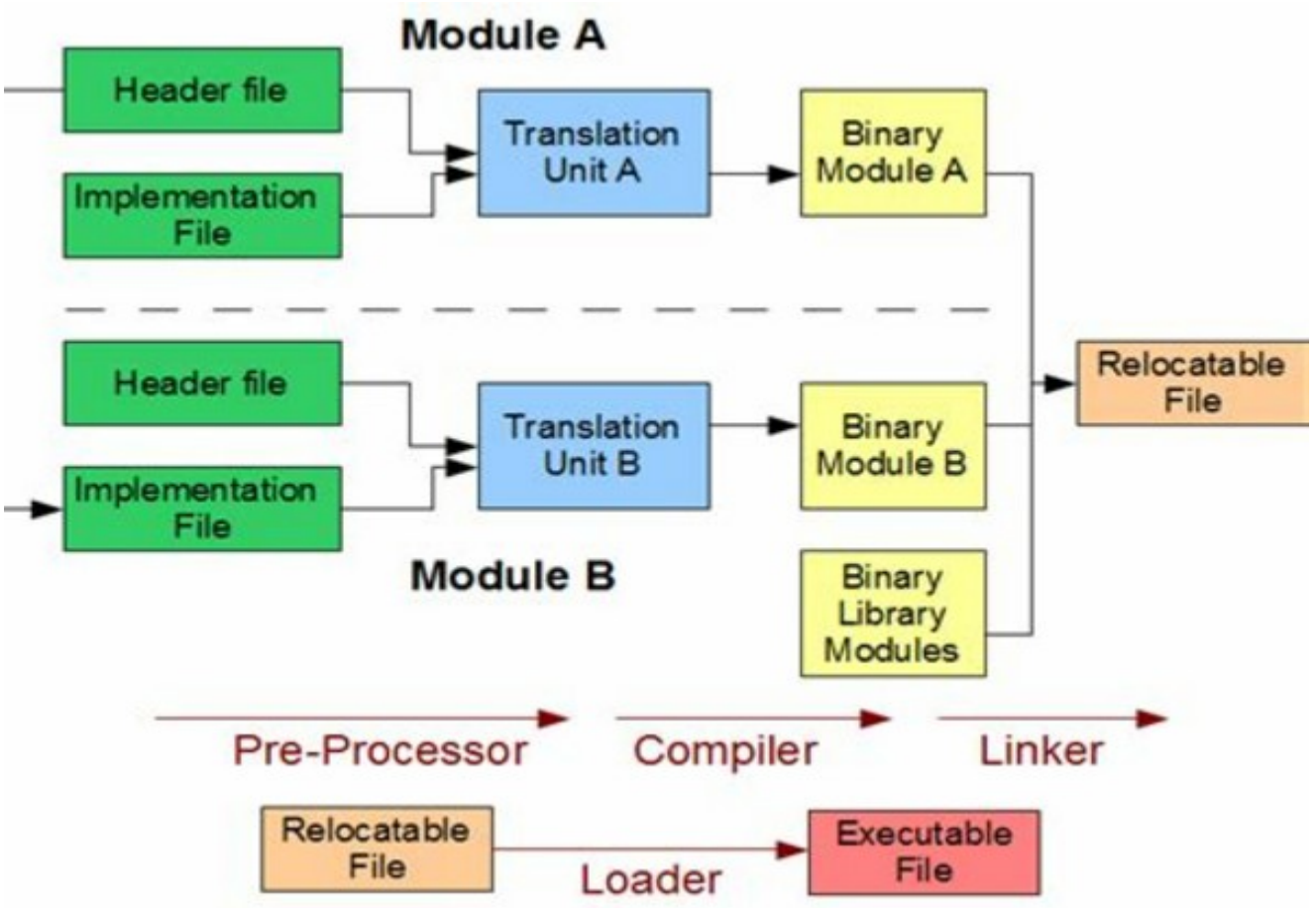
Hello

```
#include <iostream>

int main()
{
    std::cout << "Hello world" << '\n'; // std::endl
    return 0;
}

# compile + link
$ g++  hello.cpp

# execute
$ ./a.out
hello world
$
```

Hello

```
#include <iostream>

int main()
{
    std::cout << "Hello world" << '\n'; // std::endl
    return 0;
}

# compile with warnings + link + output-name
$ g++ -std=c++20 -Wextra hello.cpp -o hello

# execute
$ ./hello
hello world
$
```

C++ compilation model



Some useful compiler flags

```
# prints the precompiled source to the output
$ g++ -E hello.cpp

# compiles to assembly hello.s
$ g++ -S hello.cpp

# compiles to object hello.o, but dows not call the linker
$ g++ -c hello.cpp

# compiles all sources and then calls the linker passing hello.o too
$ g++ a.cpp b.cpp hello.o

# compiles all sources and then calls the linker passing libmath library
$ g++ a.cpp b.cpp -lmath

# compiles with optimization level
$ g++ -O2 hello.cpp

$ man g++
```

Fahrenheit

```
#include <iostream>

int main()
{
    for (int fahr = -100; fahr < 400; fahr+=10)
    {
        std::cout << "Fahr = " << fahr
                  << ", cels = " << 5/9*(fahr-32) << '\n';
    }
    return 0;
}

# compile with warnings + link + output-name
$ g++ -std=c++20 -Wextra fahr.cpp -o fahr

# execute
$ ./fahr
```

Fahrenheit

```
#include <iostream>

int main()
{
    for (int fahr = -100; fahr < 400; fahr+=10)
    {
        std::cout << "Fahr = " << fahr
                  << ", cels = " << 5/9*(fahr-32) << '\n';
    }
    return 0;
}

# compile with warnings + link + output-name
$ g++ -std=c++20 -Wextra fahr.cpp -o fahr

# execute
$ ./fahr
Fahr = -100, cels = 0
Fahr = -90, cels = 0
Fahr = -80, cels = 0
```

Fahrenheit

```
#include <iostream>

int main()
{
    for (int fahr = -100; fahr < 400; fahr+=10)
    {
        std::cout << "Fahr = " << fahr
                  << ", cels = " << 5/9*(fahr-32) << '\n';
    }
    return 0;
}

# compile with warnings + link + output-name
$ g++ -std=c++20 -Wextra fahr.cpp -o fahr

# execute
$ ./fahr
Fahr = -100, cels = 0
Fahr = -90, cels = 0
Fahr = -80, cels = 0
```

Static type system

Fahrenheit

```
#include <iostream>

int main()
{
    for (int fahr = -100; fahr < 400; fahr+=10)
    {
        std::cout << "Fahr = " << fahr
                  << ", cels = " << 5./9.*(fahr-32) << '\n';
    }
    return 0;
}

# compile with warnings + link + output-name
$ g++ -std=c++20 -Wextra fahr.cpp -o fahr

# execute
$ ./fahr
Fahr = -100, cels = -73.3333
Fahr = -90, cels = -67.7778
Fahr = -80, cels = -62.2222
```

Fahrenheit

```
#include <iostream>
#include <iomanip>   // for input-output manipulators

int main()
{
    for (int fahr = -100; fahr < 400; fahr+=10)
    {
        std::cout << "Fahr = " << std::setw(5) << fahr
                  << ", cels = " << std::fixed << setprecision(4)
                  << 5./9.*(fahr-32) << '\n';
    }
    return 0;
}

# compile with warnings + link + output-name
$ g++ -std=c++20 -Wextra fahr.cpp -o fahr

# execute
$ ./fahr
Fahr =  -100, cels = -73.3333
Fahr =    40, cels =   4.4444
Fahr =   390, cels = 198.8889
```

Fahrenheit

```
#include <iostream>
#include <iomanip>   // for input-output manipulators

const int lower = -100;
const int upper =  400;
const int step  =   10;

int main()
{
    for (int fahr = lower; fahr < upper; fahr+=step)
    {
        std::cout << "Fahr = " << std::setw(5) << fahr
                  << ", cels = " << std::fixed << setprecision(4)
                  << 5./9.*(fahr-32) << '\n';
    }
    return 0;
}
```

Fahrenheit

```
#include <iostream>
#include <iomanip>

const int lower = -100;
const int upper =  400;
const int step  =   10;

double fahr2cels( double f)  // declaration must preceed first use
{
    return 5./9.*(f-32);
}

int main()
{
    for (int fahr = lower; fahr < upper; fahr+=step)
    {
        std::cout << "Fahr = " << std::setw(5) << fahr
                  << ", cels = " << std::fixed << setprecision(4)
                  << fahr2cels(fahr) << '\n'; // int fahr converted to double
    }
    return 0;
}
```

Fahrenheit

```
#include <iostream>
#include <iomanip>

const int lower = -100;
const int upper =  400;
const int step  =   10;

double fahr2cels( double f);  // declaration must preceed first use

int main()
{
    for (int fahr = lower; fahr < upper; fahr+=step)
    {
        std::cout << "Fahr = " << std::setw(5) << fahr
                  << ", cels = " << std::fixed << setprecision(4)
                  << fahr2cels(fahr) << '\n'; // int fahr converted to double
    }
    return 0;
}

double fahr2cels( double f)  // definition of fahr2cels
{
    return 5./9.*(f-32);
}
```

Fahrenheit

```
#include <iostream>
#include <fmt/core.h>  // C++20 std::format is similar g++-13, clang++-17

const int lower = -100;
const int upper =  400;
const int step  =   10;

double fahr2cels( double f);  // declaration must preceed first use

int main()
{
    for (int fahr = lower; fahr < upper; fahr+=step)
    {
        fmt::print("Fahr = {:5d}, cels = {:.4f}\n", fahr, 5./9.*(fahr-32));
    }
    return 0;
}

double fahr2cels( double f)  // definition of fahr2cels
{
    return 5./9.*(f-32);
}
```

Fahrenheit

```
#include <iostream>
#include <fmt/core.h>  // C++20 std::format is similar g++-13, clang++-17

const int lower = -100;
const int upper =  400;
const int step  =   10;

double fahr2cels( double f);  // declaration must preceed first use

int main()
{
    for (int fahr = lower; fahr < upper; fahr+=step)
    {
        std:.string s = fmt::format("Fahr = {:5d}, cels = {:.4f}\n",
                                                    fahr, 5./9.*(fahr-32));
        std::cout << s;
    }
    return 0;
}

double fahr2cels( double f)  // definition of fahr2cels
{
    return 5./9.*(f-32);
}
```

Fahrenheit

```
#include <iostream>
#include <fmt/core.h>  // C++20 std::format is similar g++-13, clang++-17
#include <fmt/chrono.h>

int main()
{
    using namespace std::literals::chrono_literals;

    fmt::print("Default format: {} {}\n", 42s, 100ms);
    fmt::print("strftime-like format: {:%H:%M:%S}\n", 3h + 15min + 30s);
    fmt::print(fg(fmt::color::crimson) | fmt::emphasis::bold,
                                                "Hello, {}!\n", "world");
    return 0;
}
```

Program structure

- C++ program contains
  - Preprocessor directives
  - Comments
  - C++ tokens

```
/*
 My first C++ program contains a multiline comment
 */
#include <iostream>   // Preprocessor directives starts with #

int main()          // C++ tokens
{
    std::cout << "Hello world" << '\n';
    return 0;
}
```

C++ tokens

- Keywords `int` `return` `if` `for` `try` `class`
- Identifiers `main` `std` `cout` `fahr` `lower` `fahr2cels`
- Literals `-100` `400` `5.` `5.0` `0` `'\n'` `"Hello world"`
- Operators `+` `==` `<` `=` `+=` `<<` `::` `( )` `[ ]` `?:` `,`
- Separators `{` `}` `,` `;`

```
/*
 My first C++ program contains a multiline comment
 */
#include <iostream>   // Preprocessor directives starts with #

int main()          // C++ tokens
{
    std::cout << "Hello world" << '\n';
    return 0;
}
```

Identifiers

- To name variables, functions, types, ...
  - Starts with a letter A-Z a-z underscore
  - Continues with letter or digit
  - Case sensitive
  - Arbitrary long (but implementations, esp. linker may have limit)

```
i  fahr  fahr2cels   lower   If  IF  iF  if   if1

underscore_notation
camelCaseNotation
LPSZHungarianNotation

UPPERCASE_FOR_MACRO_NAMES
_reserved
```

Paper to read: http://www.cs.loyola.edu/~binkley/papers/icpc09-clouds.pdf

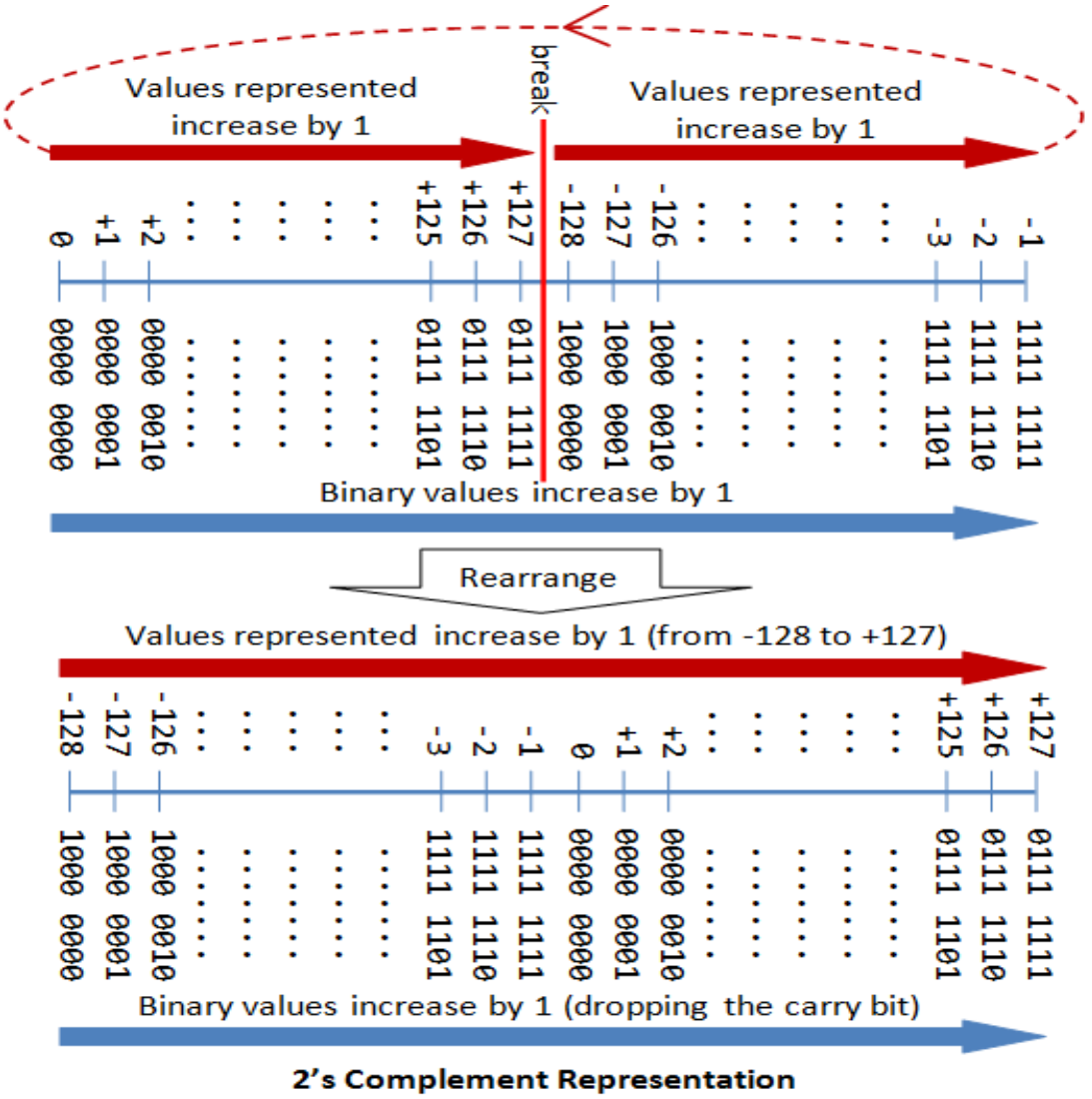
Literals

- To express values
  - Integral
  - Floating point
  - Boolean
  - Character
  - String

Integer literals

name	example	type	value
Decimal integer	25	int	25
Octal integer	031	int	25
Hexadecimal integer	0x19	int	25
Long integer	4L	long int	4
Long long integer	4LL	long long int	4
Unsigned integer	31u	unsigned int	25
Unsigned long integer	31UL	unsigned long int	25
Unsigned long hexadecimal integer	0xffffffff	unsigned long int	4294967295
Unsigned long hexadecimal integer	-1ul	unsigned long int	maybe? 18446744073709551615

Integer representation



Integer types

- Standard integer type: int, at least 16 bits, signed
  - but on modern 32/64 bit machines almost always 32 bits
  - sign modifiers: signed/unsigned
  - length modifiers: short (min 16), long (min 32), long long (min 64)
- Fixed sized integers (since C++11)
  - int8_t, int16_t, int32_t, int64_t (exactly X bits)
  - uint8_t, uint16_t, uint32_t, uint64_t (exactly X bits unsigned)
  - int_fast8_t, int_fast16_t, int_fast32_t, int_fast64_t, (fastest >=X)
  - intmax_t, uintmax_t (maximum width integers)
  - intptr_t, uintptr_t (integers capable to hold a pointer (to void))

Signed vs unsigned

- Signed integer types may overflow – undefined behavior
- Unsigned types has a modulo 2^N arithmetics
- Operations mixing signed and unsigned types may be dangerous

```
int f()
{
    unsigned ui = 1
    int       i = -2;

    if ( i + ui < 0 )  // likely false
    {
       ...
    }
    return 0;
}
```

Signed vs unsigned

```
#include <vector>
#include <iostream>
int main()
{
    std::vector<int> v = {1,2,3,4,5,6,7,8,9,10,11,12,13,14};

    for ( int i = 0; i < v.size(); ++i)   // v.size() returns the #elements
        std::cout << v[i] << ' ';

    for ( int i = v.size()-1; i >= 0; --i)
        std::cout << v[i] << ' ';

    return 0;
}
```

Signed vs unsigned

```
#include <vector>
#include <iostream>
int main()
{
    std::vector<int> v = {1,2,3,4,5,6,7,8,9,10,11,12,13,14};

    for ( int i = 0; i < v.size(); ++i)   // v.size() returns the #elements
        std::cout << v[i] << ' ';

    for ( int i = v.size()-1; i >= 0; --i)
        std::cout << v[i] << ' ';

    return 0;
}

$ g++ -Wextra p.cpp
usize.cpp: In function ‘int main()’:
usize.cpp:7:28: warning: comparison of integer expressions of different
signedness: ‘int’ and ‘std::vector<int>::size_type’ {aka ‘long unsigned int’}
[-Wsign-compare]
    7 |         for ( int i = 0; i < v.size(); ++i)
      |                          ~~^~~~~~~~~~
```

Signed vs unsigned

```
#include <vector>
#include <iostream>
int main()
{
    std::vector<int> v = {1,2,3,4,5,6,7,8,9,10,11,12,13,14};

    for ( std::size_t i = 0; i < v.size(); ++i)   // std::size_t is unsigned
        std::cout << v[i] << ' ';

    for ( std::size_t i = v.size()-1; i >= 0; --i)
        std::cout << v[i] << ' ';

    return 0;
}

$ g++ -Wextra p.cpp
$
```

Signed vs unsigned

```
#include <vector>
#include <iostream>
int main()
{
    std::vector<int> v = {1,2,3,4,5,6,7,8,9,10,11,12,13,14};

    for ( std::size_t i = 0; i < v.size(); ++i)   // std::size_t is unsigned
        std::cout << v[i] << ' ';

    for ( std::size_t i = v.size()-1; i >= 0; --i)
        std::cout << v[i] << ' ';

    return 0;
}

$ g++ -Wextra p.cpp
$ ./a.out  #ooops
```

Signed vs unsigned

```
#include <vector>
#include <iostream>
int main()
{
    std::vector<int> v = {1,2,3,4,5,6,7,8,9,10,11,12,13,14};

    for ( ptrdiff_t i = 0; i < std::ssize(v); ++i)  // ssize() returns signed
        std::cout << v[i] << ' ';

    for ( ptrdiff_t i = std::ssize(v)-1; i >= 0; --i)
        std::cout << v[i] << ' ';

    return 0;
}

$ g++ -Wextra p.cpp
$ ./a.out  # works fine
1 2 3 4 5 6 7 8 9 10 11 12 13 14 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

Signed vs unsigned

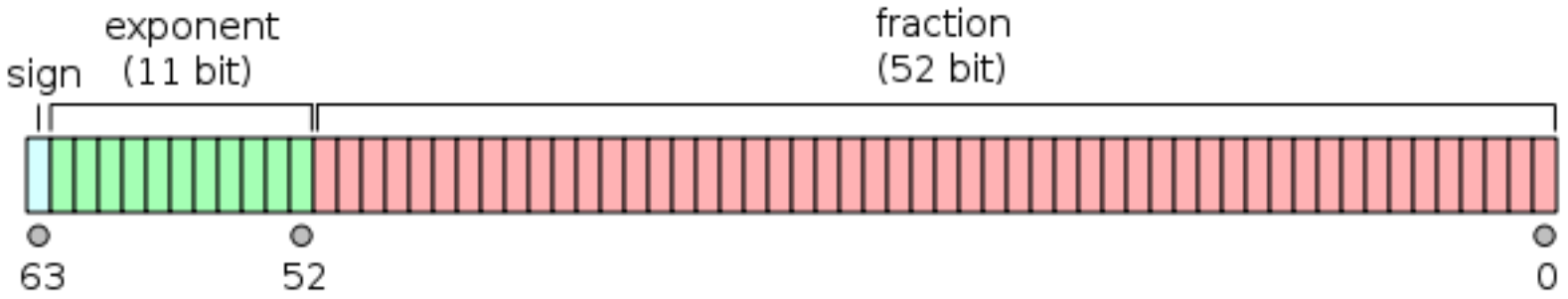
```
#include <vector>
#include <iostream>
int main()
{
    std::vector<int> v = {1,2,3,4,5,6,7,8,9,10,11,12,13,14};

    for ( auto i = 0; i < std::ssize(v); ++i)     // i will be integer
        std::cout << v[i] << ' ';

    for ( auto i = std::ssize(v)-1; i >= 0; --i)  // i will be ptrdiff_t
        std::cout << v[i] << ' ';

    return 0;
}

$ g++ -Wextra p.cpp
$ ./a.out  # works fine
1 2 3 4 5 6 7 8 9 10 11 12 13 14 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

Floating point literals

name	example	type	value
Double literal	3.14	double	3.14
Float literal	3.14F	float	3.14
Long double literal	3.14L	long double	3.14
Double literal	3.14e3	double	3140.0 (3.14x10^3)
Long double literal	-3.14E-2L	long double	-0.0314 -(3.14x10^(-2))
Double literal	1’000’000.	double	1,000,000.0 (‘ are ignored) (C++17)
Hexadecimal floating point literal	0x1.4p3	double	10.0 (1.25)x(2^3) (C++17)

IEEE754

C++ type	precision	#bits	sign	#exponent bits	#fragment bits
float	single	32	1	23	8
double	double	64	1	52	11
n/a	extended	80	1	64	15
long double	quadrouple	128	1	112	15



Character literals

name	example	type	numeric value
Character literal	`'A'`	char	65
Escape sequence	`'\n'`	char	10
Escape sequence	`'\''`	char	Asci value of ‘
Octal escape sequence	`'\377'`	char	255
Hexadecimal escape sequence	`'\xff'`	char	255
UTF-8 character literal	`u8'A'`	char8_t (C++20)	65
UTF-16 character literal	`u'猫'`	char16_t	4
UTF-32 character literal	`U'猫'`	char32_t	25
Wide character literal	`L'β'`	wchar_t	25
Multicharacter literal (optional)	`'AB'`	int	Implementation defined
Wide multicharacter literal (optional)	`L'AB'`	unsigned long int	maybe? 18446744073709551615

Fundamental types summary

- Integers
  - int is the default and “best”
- Boolean true ==> 1 false ==> 0
- Byte (C++17)
  - `enum class byte : unsigned char {};`
- Char also an “integral type” (min 8 bits)
  - char/signed char/unsigned char: 2 of them are equivalent
  - wchar_t for wide characters
- Floating point types
  - double is the default and “best”

String literals

name	example	type
Ordinary string literal	`"Hello"`	const char [N]
Wide string literal	`L"Hello"`	const wchar_t [N]
Ordinary string literal	`"Hel\xfflo\n"`	const char [N]
UTF-8 string literal	`u8"Hello\n"`	const char8_t [N]
UTF-16 string literal	`u"Hell猫\n"`	const char16_t [N]
UTF-32 string literal	`U"Hell猫\n"`	const char32_t [N]

Raw string literal	`R"begin(`
	`  Hello`
	`world`
	`)\nend”`
		const char [N]
		
		“begin\n(Hello\nworld\n)\\nend”

String literals

- Constant array of the corresponding characters
  - Containing the terminator ‘\0’
  - Attempt to change them is undefined behavior (run-time error)
- Static storage (lives from the beginning to the end of the program)
- Storage may optimized to reuse (sub) string literals
- Adjacent string literals are concatenated

```
"Hello " "world" == "Hello world"

 3+"Hello" =?= "lo"

sizeof("Hello") == 6
```

sizeof()

```
//
// This is _very_ compiler/platform specific
//
#include <iostream>

int main()
{
  std::cout << sizeof(char) << " ";
  std::cout << sizeof(bool) << " ";
  std::cout << sizeof(short) << " ";
  std::cout << sizeof(int) << " ";
  std::cout << sizeof(long) << " ";
  std::cout << sizeof(long long) << " ";
  std::cout << sizeof(float) << " ";
  std::cout << sizeof(double) << " ";
  std::cout << sizeof(long double) << " ";
  std::cout << sizeof(int*) << " ";
  std::cout << sizeof("Hello world") << "\n";

  return 0;
}
```