

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

C++

C++ intro

Mike Burrell

February 27, 2024

Readings for this set of slides

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

■ None

Learning objectives

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- Understand high-level mechanisms of how a computer works
- See an overview of how programming languages developed
- Get comfortable with a C++ development environment

Machine code

C++

Mike Burrell

Readings

History

c

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- Before we talk about C++, we need to talk about C
- Before we talk about C, we need to talk about machine code
- Every computer has an *architecture*
 - PCs (and older Macs) have an architecture of AMD64T, also known as x86-64
 - Newer Macs (M1 or M2) have an architecture of ARM64
 - Smartphones are generally ARM64 as well
 - There are other architectures in use today (POWER, RISC-V, ...)
- Every architecture specifies a different *instruction set*, and requires different *machine code* to accomplish the same tasks

Example machine code (Fibonacci function)

C++

Mike Burrell

Readings

History

c

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

AMD64T

```
fib:    xorl    %edx, %edx
        movl    $1, %ecx
        xorl    %eax, %eax
loop:   cmpq    %rdi, %rdx
        jge     done
        leaq    (%rax,%rcx), %rsi
        incq    %rdx
        movq    %rax, %rcx
        movq    %rsi, %rax
        jmp     loop
done:   ret
```

Example machine code (Fibonacci function)

C++

Mike Burrell

Readings

History

c

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

ARM64

```
fib:    mov     x1, x0
        mov     x2, 0
        mov     x3, 1
        mov     x0, 0
loop:   cmp     x2, x1
bge     done
        add     x4, x0, x3
        add     x2, x2, 1
        mov     x3, x0
        mov     x0, x4
        b       loop
done:   ret
```



C

C++

Mike Burrell

Readings

History

c

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- C came about in 1969/1970 as a way to portably write Unix
 - Bell Labs had a new operating system (Unix) that they wanted run on different hardware
 - Creating a program language seemed better than rewriting everything in assembly for a new architecture
- C is fundamentally a system programming language
 - Its value is writing machine code systems software (operating systems, system utilities, etc.) in a portable way
 - It is sometimes called “the portable assembler”

C

C++

Mike Burrell

Readings

History

c

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- C is balancing (quite well) between two objectives
 - Be very low-level and expose access to the underlying hardware
 - Be portable and abstract away any differences between hardware

C

C++

Mike Burrell

Readings

History

c

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- C is balancing (quite well) between two objectives
 - Be very low-level and expose access to the underlying hardware
 - Be portable and abstract away any differences between hardware
- There are some times when we think of something abstractly, but C allows mechanisms to see it concretely, as the machine does
- All of this applies to C++, as well

Object-oriented design

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- At about the same time C was being developed, a new idea sprouted of research labs
 - It was called *object-oriented design*
 - It led to two important programming languages called Smalltalk and Simula
- Smalltalk and Simula had *objects* and *classes* which allowed for complex software to be written in a clear way
 - This was difficult to do in C!

Inspirations from Lisp

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- Smalltalk, in particular, went beyond just inventing object-oriented programming
- It took inspiration from earlier *functional* programming languages like Lisp
- It used a style of programming called *metaprogramming*
 - Metaprogramming is writing a program which writes a program
 - Metaprogramming allowed a lot of power for writing very abstract code

Merging it together

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

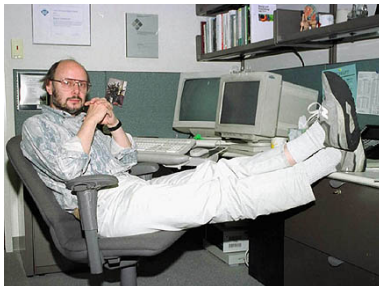
Integers

sizeof

Signedness

Literals

Conclusion



- In 1979, a Bell Labs technician named Bjarne Stroustrup started creating a new programming language called “C With Classes”
- He wanted a low-level portable systems language (like C) with object-oriented design and metaprogramming (like Smalltalk and Simula)

C++ and snowballing

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- While developing “C With Classes” (soon renamed C++), a lot of people came to Stroustrup
 - “Can you put in exceptions?”
 - “Can you put in multiple inheritance?”
 - “Can you put in metaprogramming?”
 - “Can you put in ...?”
- Infamously (said by other members at Bell Labs), Stroustrup “couldn’t say ‘no’”
- Right from the beginning, C++ was language which tried to include every feature
 - Today it stands as arguably the most complicated programming language ever made

Timeline since then

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

C++98/03 — standardized by ANSI and ISO

C++11/14 — (**we will focus on this for the course**)
stronger compatibility with C, type
inference, for-each loops, lambdas
(functional programming), smart pointers

C++17 — mostly syntax cleanups and library
additions

C++20 — metaprogramming features, functional
programming features

C++23 — (**allowed to be used in your
assignments**) big changes to
metaprogramming

C++26 — not done yet

Brief wrapup of C++

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- C++ is a portable, low-level systems language
- It incorporates many different programming paradigms (procedural, functional, object-oriented, metaprogramming, etc.)
- It is one of the most complex languages ever made
 - It is not possible to learn (all of) C++ in one course
 - It is not possible to learn (all of) C++ in 10 years
- The first few months of this course will be learning C++98, then C++11/C++14
- Officially we are learning C++23 (and you can use it in your assignments), but we do not have time to cover features beyond C++11 in detail

Typing

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- Unlike Python (but like other languages, like Java), C++ is *statically typed*
- This means every variable must be *declared* with a *type* before it is given a value
- E.g., `int x = 3;` says to declare a new variable `x` of type integer and give it the value 3
- Choosing the appropriate type is important

Integer widths

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- One important example of where C and C++ are both low-level and abstract is with data types
- `char`, `short`, `int`, `long`, `long long`, `float`, `double` and `long double` are all quite loosely defined

Integer widths

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- One important example of where C and C++ are both low-level and abstract is with data types
- `char`, `short`, `int`, `long`, `long long`, `float`, `double` and `long double` are all quite loosely defined

Type	Guarantees in C, C++
------	----------------------

<code>char</code>	At least 8 bits in size, size is 1
-------------------	------------------------------------

Integer widths

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- One important example of where C and C++ are both low-level and abstract is with data types
- char, short, int, long, long long, float, double and long double are all quite loosely defined

Type

Guarantees in C, C++

char

At least 8 bits in size, size is 1

short

At least 16 bits, not smaller than char, not bigger than int

Integer widths

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- One important example of where C and C++ are both low-level and abstract is with data types
- `char`, `short`, `int`, `long`, `long long`, `float`, `double` and `long double` are all quite loosely defined

Type	Guarantees in C, C++
<code>char</code>	At least 8 bits in size, size is 1
<code>short</code>	At least 16 bits, not smaller than <code>char</code> , not bigger than <code>int</code>
<code>int</code>	At least 16 bits, a “natural size” of the machine

<code>char</code>	At least 8 bits in size, size is 1
<code>short</code>	At least 16 bits, not smaller than <code>char</code> , not bigger than <code>int</code>
<code>int</code>	At least 16 bits, a “natural size” of the machine

Integer widths

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- One important example of where C and C++ are both low-level and abstract is with data types
- char, short, int, long, long long, float, double and long double are all quite loosely defined

Type	Guarantees in C, C++
char	At least 8 bits in size, size is 1
short	At least 16 bits, not smaller than char, not bigger than int
int	At least 16 bits, a “natural size” of the machine
long	At least 32 bits, not smaller than int

Integer widths

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- One important example of where C and C++ are both low-level and abstract is with data types
- `char`, `short`, `int`, `long`, `long long`, `float`, `double` and `long double` are all quite loosely defined

Type	Guarantees in C, C++
<code>char</code>	At least 8 bits in size, size is 1
<code>short</code>	At least 16 bits, not smaller than <code>char</code> , not bigger than <code>int</code>
<code>int</code>	At least 16 bits, a “natural size” of the machine
<code>long</code>	At least 32 bits, not smaller than <code>int</code>
<code>long long</code>	At least 64 bits, not smaller than <code>long</code>

Basic integer types

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- This means different machines (and different compilers) are free to defined types in different ways
- On a PDP-11 and 16-bit x86, 8-bit byte, 16-bit short, 16-bit int, 32-bit long
- On a CDC 6600, 18-bit byte, 18-bit short, 80-bit int, 80-bit long
- On x86-64 Windows, 8-bit byte, 16-bit short, 32-bit int, 32-bit long
- On x86-64 Linux, 8-bit byte, 16-bit short, 32-bit int, 64-bit long
- On SPARC64 Solaris, 8-bit byte, 16-bit short, 64-bit int, 64-bit long
- On UNICOS, 8-bit byte, 64-bit short, 64-bit int, 64-bit long

How do we write portable code?

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- If every compiler makes the data types different, how can our code be portable?
- Don't make unnecessary assumptions
 - Just because `int` is 32-bit on your computer, assume it might be 16-bits (or 80-bits) on someone else's
 - Almost never will you have to rely on a variable being of a specific width
- Also, there are more types that are defined for us. . . .

size_t

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- `size_t` is a very important type in C++ and we will see it a *lot*
- `size_t` is an alias for an integer type (probably either `unsigned int`, `unsigned long` or `unsigned long long`)
- It is used to represent:
 - The size of an object (the number of bytes of memory consumed)
 - The number of elements in an array
 - An index into an array
- It will 32 bits in size on 32-bit architectures and 64 bits in size on 64-bit architectures

size_t in action

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

```
1 string x = "hello";
2 size_t n = x.length(); // use a size_t for lengths!
3 for (size_t i = 0; i < n; i++) { // and for indices!!
4     cout << x[i] << endl;
5 }
```

Don't use int needlessly!

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

If you use `int` for sizes/indices:

- Your code will not work on large arrays on 64-bit platforms!!
 - On most 64-bit platforms, `int` will still only be 32 bits
 - `size_t` is the only type which is guaranteed to be the correct size to represent a size/length/index on any platform!
 - `size_t` will be 32 bits on 32-bit platforms and 64 bits on 64-bit platforms

More info: [https:](https://en.cppreference.com/w/cpp/types/size_t)

[//en.cppreference.com/w/cpp/types/size_t](https://en.cppreference.com/w/cpp/types/size_t)

sizeof

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers


sizeof

Signedness

Literals

Conclusion

- C and C++ have a unary operator¹ defined called `sizeof`
- The operand to `sizeof` may be:
 - A type name; or
 - A value (such as a variable)
- `sizeof` returns a value of type `size_t` which is measured in *characters* (*bytes*)
 - Remember that the size of `char` *must* be 1

¹A *unary operator* is an operator with 1 operand. 

climits

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

There is a file called `climits` we can define which includes a number of constants which are sometimes helpful:

- `CHAR_BIT` — the number of bits in a byte (usually 8)
- `INT_MIN` — -32768 on 16-bit machines, -2147483648 on 32-bit (and many 64-bit) machines
- `INT_MAX` — +32767 on 16-bit machines, +2147483647 on 32-bit/64-bit machines

Similarly, `SHORT_MIN`, `SHORT_MAX`, `LONG_MIN`, `LONG_MAX`, etc.

Signedness

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- Every integer type in C and C++ comes in two flavours: signed and unsigned
- Signed integers can represent negative numbers
- Unsigned integers cannot represent negative numbers
- Signed integers are stored in 2's complement
- Integers are signed by default
- Weird things happen when an integer value overflows, so we will choose integer types big enough to avoid overflows

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      signed int x = -5; // OK
8      unsigned int y = -5; // NOT OK
9      unsigned char z = 200; // OK
10     signed char w = 200; // NOT OK
11     int v = w; // int is the same as signed int
12     return 0;
13 }
```

Literals

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- Most literals work similarly to in other languages like Python
 - E.g., 15 is a literal of type `int`, 3.5 is a literal of type `double`
- String literals "like this" are actually of type `char []`, not `string`, but can be implicitly converted to `string`
- C has literal syntax beyond what Java has, too

Integer literals

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

	Type
5, -145	int
5L, -145L	long
5LL, -145LL	long long
5U, 1327U	unsigned int
5UL, 1327UL	unsigned long
5ULL, 1327ULL	unsigned long long
0134	int (given in base 8)
0x134	int (given in base 16)
0b1011	int (given in base 2)

By convention, we usually use uppercase L for long literals because l (lowercase) can be easily confused for 1 (the numeral one). The C++ language allows either uppercase or lowercase suffixes, however.

Floating-point literals

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

	Type
3.0, -4347.271	double
3.0f, -4347.271f	float
3.0L, -4347.2471L	long double
37.165e12, 6e-4	double in scientific notation
0x23p9	double in scientific notation with mantissa in base 16

Types in C++

C++

Mike Burrell

Readings

History

C

Smalltalk and Simula

Conclusion

Data types

Typing

Integers

sizeof

Signedness

Literals

Conclusion

- C++ integer sizes vary from platform to platform, so don't make unnecessary assumptions
- Use `size_t` for lengths, sizes and indices
- C++ has unsigned integer types as well as signed
- `sizeof (char)` is 1, and `sizeof x` will tell us how many bytes are needed to store `x`