Variables

2025 Winter APS105: Computer Fundamentals
Jon Eyolfson

Lecture 2
1.0.1

Programs Always Access Memory

When we talk about memory, we're referring to RAM (also called main memory)

We can access storage ("long-term" memory) through main memory by asking the OS (not covered in this course)

Computers Access Memory in Bytes

Recall: 1 byte is 8 bits

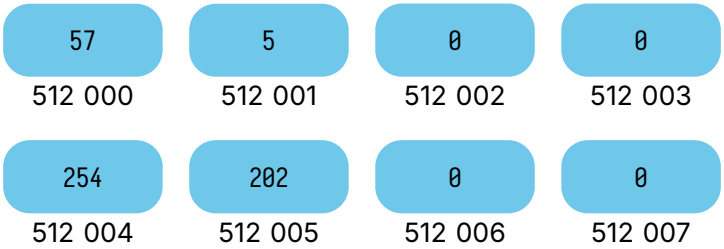
There are 256 different representations (2^8)

If we represent a byte as a whole number,
a byte can represent 0 to 255 (inclusive)

In general if we use n bits to represent a whole number,
it can represent 0 to $2^n - 1$ (inclusive)

Every byte has its own unique address (location)
We call memory byte addressable

Each Address Contains a Byte (Value in Blue, Address Below)



Your CPU Has Instructions to Access Memory

An instruction can read a byte from memory
(retrieving the value from memory)

Another instruction can write a byte to memory
(sending the value to memory)

Thankfully, C handles this for us

The Number of Possible Addresses Depends on Your CPU

A 64-bit CPU means that an address can be up to 64 bits

There are up to 2^{64} addresses, or 18 446 744 073 709 551 616
That's up to 18 quintillion bytes!

Computers can hold a lot of bytes, we communicate using prefixes instead

The Typical Prefixes are Kilo, Mega, and Giga

We write byte as a unit as “B”, and a bit as a “b”

Since computers are binary machines, we use powers of 2 for prefixes

1 KB (kilobyte) is 1024 bytes, or 2^{10} bytes

1 MB (megabyte) is 1024 kilobytes, or 2^{20} bytes

1 GB (gigabyte) is 1024 megabytes, or 2^{30} bytes

Old Computers Had 32-Bit CPUs

Since memory is byte addressable that means it can only access 2^{32} bytes

If we want to use our prefixes, we can re-write 2^{32} as: $2^2 2^{30}$ bytes
or in other words, it can only use 4 GB of memory

C Uses Types to Describe Values

Types tell you two things:

1. What they represent
2. How much memory they use

An `int` is a type that represents an integer and uses 4 bytes (positive and negative numbers)

How Do We Represent An Integer?

Again, the type for an integer in C is `int`
  It's 4 bytes, which is 32 bits (2^{32} different representations)

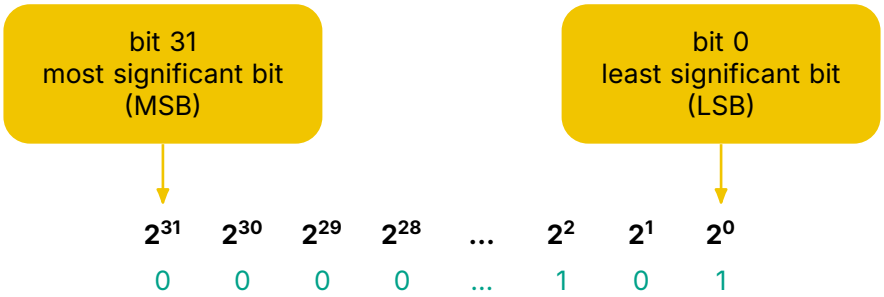
If this was a whole number what numbers could it represent?

How Do We Represent An Integer?

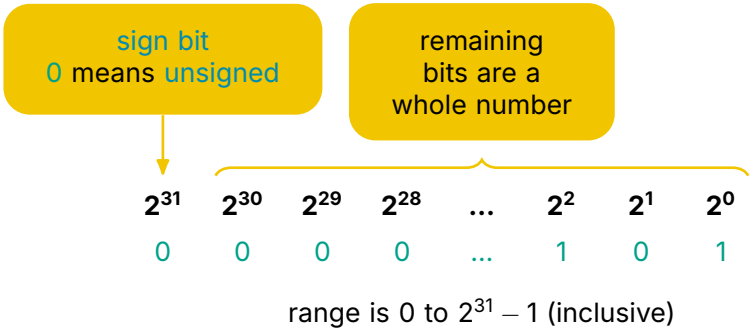
Again, the type for an integer in C is `int`
  It's 4 bytes, which is 32 bits (2^{32} different representations)

If this was a whole number what numbers could it represent?
  A number between 0 and 4 294 967 295 (inclusive)

A C `int` Represents an Integer Between -2^{31} and $2^{31} - 1$

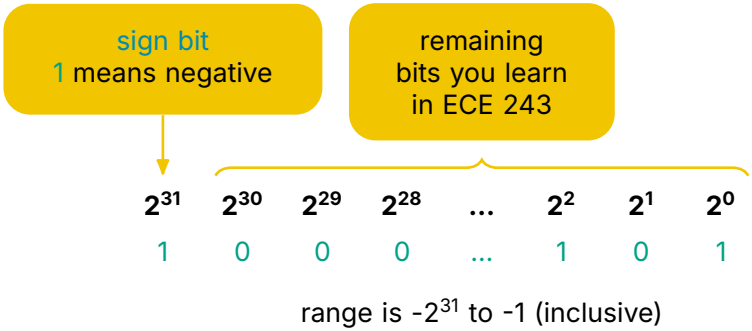


A C `int` Represents an Integer Between -2^{31} and $2^{31} - 1$



range is 0 to $2^{31} - 1$ (inclusive)

A C `int` Represents an Integer Between -2^{31} and $2^{31} - 1$



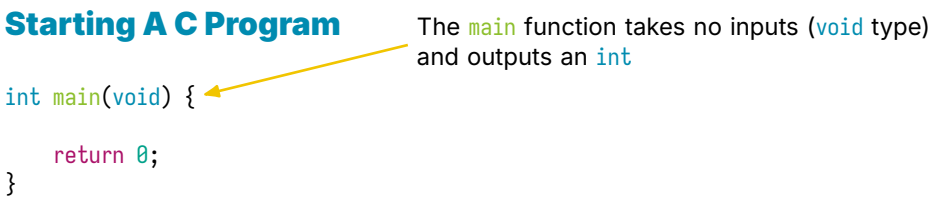
range is -2^{31} to -1 (inclusive)

Starting A C Program

```c
int main(void) {

    return 0;
}
```

Starting A C Program



Starting A C Program

```
int main(void) {

    return 0;
}
```

The `main` function takes no inputs (`void` type) and outputs an `int`

We can add statements between the { and } that the computer runs, in order

Starting A C Program

```
int main(void) {

    return 0;
}
```

The `main` function takes no inputs (`void` type) and outputs an `int`

We can add statements between the { and } that the computer runs, in order

A return statement starts with `return` then a value that matches function's output type (the function outputs the value and finishes)

Starting A C Program

```
int main(void) {

    return 0;
}
```

The `main` function takes no inputs (`void` type) and outputs an `int`

We can add statements between the { and } that the computer runs, in order

A return statement starts with `return` then a value that matches function's output type (the function outputs the value and finishes)

Note that usually every statement ends with a `;` in C

You Can Declare (Create) Variables in C

In programming languages, there's a syntax (set of rules) you have to follow

The syntax for declaring variables is:

`<type> <name>;`

Where you replace:

`<type>` by a C type (we just know `int` for now), and
`<name>` by a name that starts with a letter followed by letters, digits, and underscores (no spaces)

We Can Declare A Variable Called x

```
int main(void) {
    int x;

    return 0;
}
```

We Can Declare A Variable Called x

```
int main(void) {
    int x;

    return 0;
}
```

We Can Declare A Variable Called x

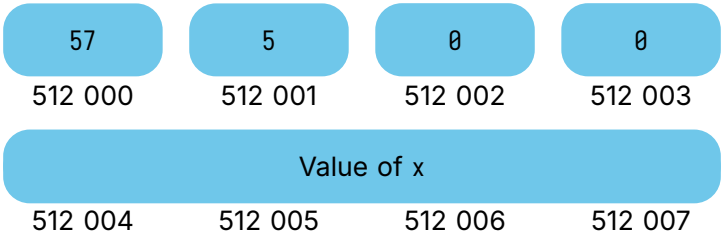
```
int main(void) {
    int x;

    return 0;
}
```

We declare an integer called x

After executing the declaration statement,
C sets aside 4 bytes to use for x
(we can use x to get its value)

C Stores the Value of `x` Somewhere in Memory



We Can Change the Value of a Variable through Assignment

The syntax of an assignment statement is:

```
<name> = <value>;
```

Where you replace:

`<name>` by a name that you previously declared

`<value>` by a value that is representable by the declared type of `<name>`

Note, unlike mathematics we can re-assign the value of a variable

Let's Change the Value of x

```
int main(void) {
    int x;

    x = 1;

    return 0;
}
```

Let's Change the Value of x

```
int main(void) {
    int x;

    x = 1;

    return 0;
}
```

Let's Change the Value of x

```c
int main(void) {
    int x;
        // At this line x has a random value
    x = 1;
        // Now, x has a value of 1
    return 0;
}
```

We Can Do Assignment as Part of a Variable Declaration

Other syntax we can use is:

```
<type> <name> = <value>;
```

This statement creates a variable and assigns an initial value
(an initial value is the first value the variable has)

We also call this initialization

A C `char` Represents a Character

A character is letters, digits, punctuation, whitespace, and control characters (whitespace is spaces, tab characters, and return characters)

You can't see control characters on your screen, but they do something (e.g., tab and return)

A `char` is always 1 byte

`char` Values Start and End with `'`

A `char` is only a single character, an example declaration is:

```
char c = 'A';
```

We need to represent invisible characters (called non-printable characters)

We prefix them with \ followed by another character we need to look up (this is called an escape character, telling C we're breaking the normal rules)

For example (see Wikipedia for more):

```
char tab = '\t';
char newline = '\n'; (this is the return character)
char backslash = '\\';
char singleQuote = '\'';
```

We Can Also Interpret a `char` as a Whole Number

Recall a byte can represent whole numbers in the range 0 to 255

Which number corresponds to which character is defined by ASCII
(American Standard Code for Information Interchange)

ASCII only uses 7 bits (the most significant bit of the byte is always 0)
  Recall: this corresponds to whole numbers in the range 0 to 127

You can look up the values on `https://www.asciitable.com/`
  (printing non-English characters is out of scope for this course)

A C `double` Represents a Number with a Decimal Point

If a number doesn't require a decimal point you should use `int`

A `double` is 8 bytes
(there's also a less precise option called a `float` that's 4 bytes)

You'll learn about how we represent these in binary in later courses

Example:

```
double pi = 3.14159265358979323846;
```

Our Last Type is a `bool`

`bool` is short for boolean which can only be `true` or `false`

It may only take up 1 bit (in practice it takes up 1 byte)
(in some situations it may lead to a faster program, use it if you can)

By default, you can't use the `bool` type, you need to add:

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

to the beginning of your code

Example:

```
bool isSnowing = true;
```

All Previous Type (Except for `bool`) are Enabled by Default

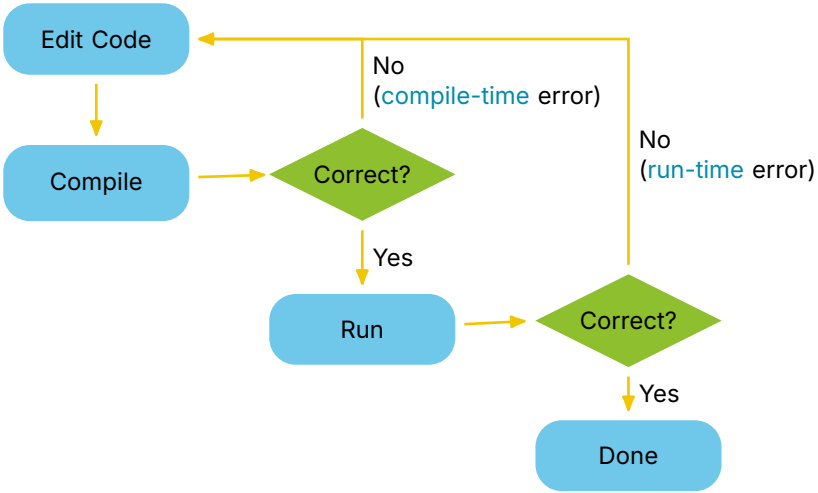
The default (built-in) types are called primitive types

For now, we'll only use: `int`, `char`, `double`, and `bool`

There are other built-in types (we'll see some more soon)

We'll also create our own types later!

Our Usual Software Development Cycle



Compile-time Errors Mean We Can't Create a Program

If we try to compile:

```
int main(void) {
    x = 1;
    int x;
    return 0;
}
```

We'll see:

```
undefined-x.c:2:5: error: use of undeclared identifier 'x'
x = 1;
^
1 error generated.
```

C Runs Line By Line from the Start of `main`

```
undefined-x.c:2:5: error: use of undeclared identifier 'x'
```

`undefined-x.c` is the name of the C source code file (`.c` extension is for C)
- The first number after the filename is the line of the error
- The second number after the filename is the column of the error

`error` means this problem at the location above is preventing compilation

This error means we cannot assign a value to a name that does not exist
- Identifier is another word for a name, in this case the name is `x`