Computer Code

PHYS2G03

McMaster University

hello.cpp
A source file with code in C++

```
#include <iostream>

int main()
{
  std::cout << "Hello
    World!\n";
}
```

We will use
code written in C++

hello.cpp

to illustrate generic
aspects of all
computer code

Anatomy of hello.cpp

```
#include <iostream>        header declares std::cout

int main() {
  std::cout << "Hello World!\n";     Output text to std output
                                     (the terminal)
}
```

Headers are mainly for the compiler – to declare things the programmer did not, like std::cout

Lines beginning with # are pre-processor directives

No header program helno.cpp

```
[wadsley@phys-ugrad ~]$ more helno.cpp
int main()
{
  std::cout << "Hello world?\n";
}

[wadsley@phys-ugrad ~]$ c++ helno.cpp -o helno
helno.cpp: In function ‘int main()’:
helno.cpp:3:3: error: ‘cout’ is not a member of ‘std’
   std::cout << "Hello world?\n";
   ^
[wadsley@phys-ugrad ~]$
```

Without the #include, the compiler is confused when I use std::cout

#include <iostream> gives us std::cout and std::cin (among other things)

Anatomy of hello.cpp

```
#include <iostream>          header declares std::cout

int main()                   Main function
{
  std::cout << "Hello World!\n";    Output text to std output
                                    (the terminal)
}
```

The curly brackets (also called braces) **{ }** enclose the source code that belongs to something – in this case the Main program

Compare
C vs. C++

hello.c
A basic C source file

```
#include <stdio.h>


int main()
{
  printf( "Hello World!\n” );
}
```

Since C++ is a superset/extension of C this is also valid C++ code

c++ will compile this too

printf is the C way of printing
declared in stdio.h

Anatomy of a C/C++ source file with a main program

#include <header> Header tells compiler where to find code not explicitly defined here

int main(arguments) main function → every program needs one

Start here when program runs

{

actual code; actually do something

more code;

return 0; return to the operating system (happens automatically at the end of the main function)

}

Anatomy of a C/C++ source file with a main program

#include <header> Header tells compiler where to find code not explicitly defined here

int main(arguments) main function → every program needs one

int means promise to return an integer

{

actual code; actually do something

more code;

return 0; return to the operating system

Provide the promised integer (e.g. 0)

}

Anatomy of a C/C++ source file with a main program

Why does it look like this?

```
#include <header>  Header tells compiler where to find
                   code not explicitly defined here
int main(arguments)      Main function
                   Start here when program runs
{
   actual code;      actually do something
   more code;

   return 0;         return to the OS
}
```

C was designed to write Unix commands like ls

The idea is that Unix

- Starts the program at the main function
- gives it the command line arguments (optional) e.g. printf hello! the first argument is “hello!”
- the program runs
- returns 0 if successful and some other integer if it failed (optional)

Anatomy of hello.cpp

```
#include <iostream>


int main()    main function:
                  promised an integer return
{
  std::cout << "Hello World!\n";
}                 end of main – automatically exits
              program and returns to prompt
```

Note: Compiler turns a blind eye if main fails to return the promised integer

Anatomy of generic C/C++ source file

```
#include <header1>
#include <header2>
…
int myfunction()
 {
   int x;
   x = 5 + 10;
   …
   return x;
}
```

Anatomy of generic C/C++ source file

```
#include <header1>
#include <header2>
…
int myfunction()
 {
   int x;
   x = 5 + 10;
   …
   return x;
}
```

Headers

A function: name chosen by programmer (promise an int)

variable declaration integer x

actual code to do something
Each line ends in ;

return value of function (int) an integer x

Anatomy of generic C/C++ source file

```
#include <header1>
#include <header2>
…
int myfunction()
 {
   int x;
   x = 5 + 10;
   …
   return x;
}
```

NOTE: This source file has no main function so it cannot make a program by itself.

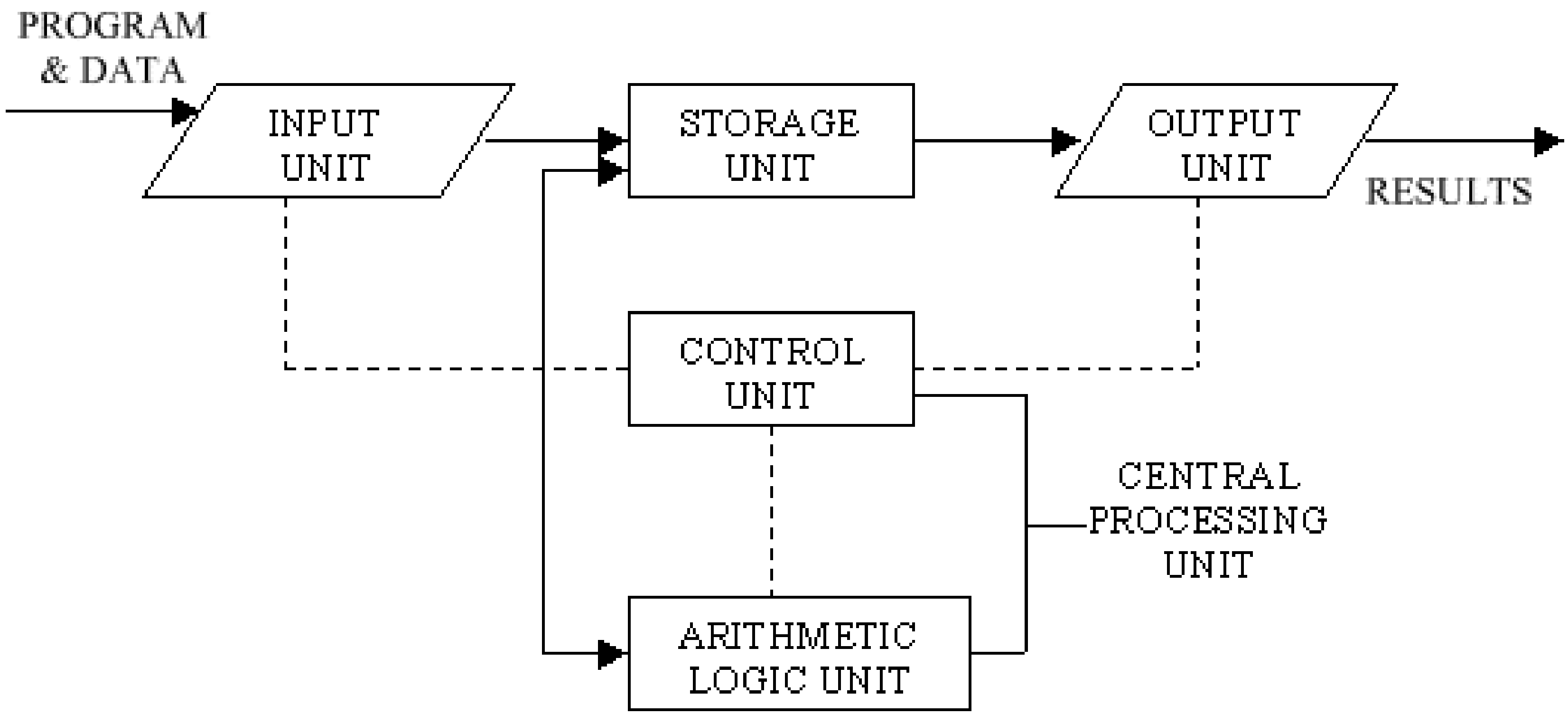
Every program needs a main function somewhere to define the starting point.

What is a computer?

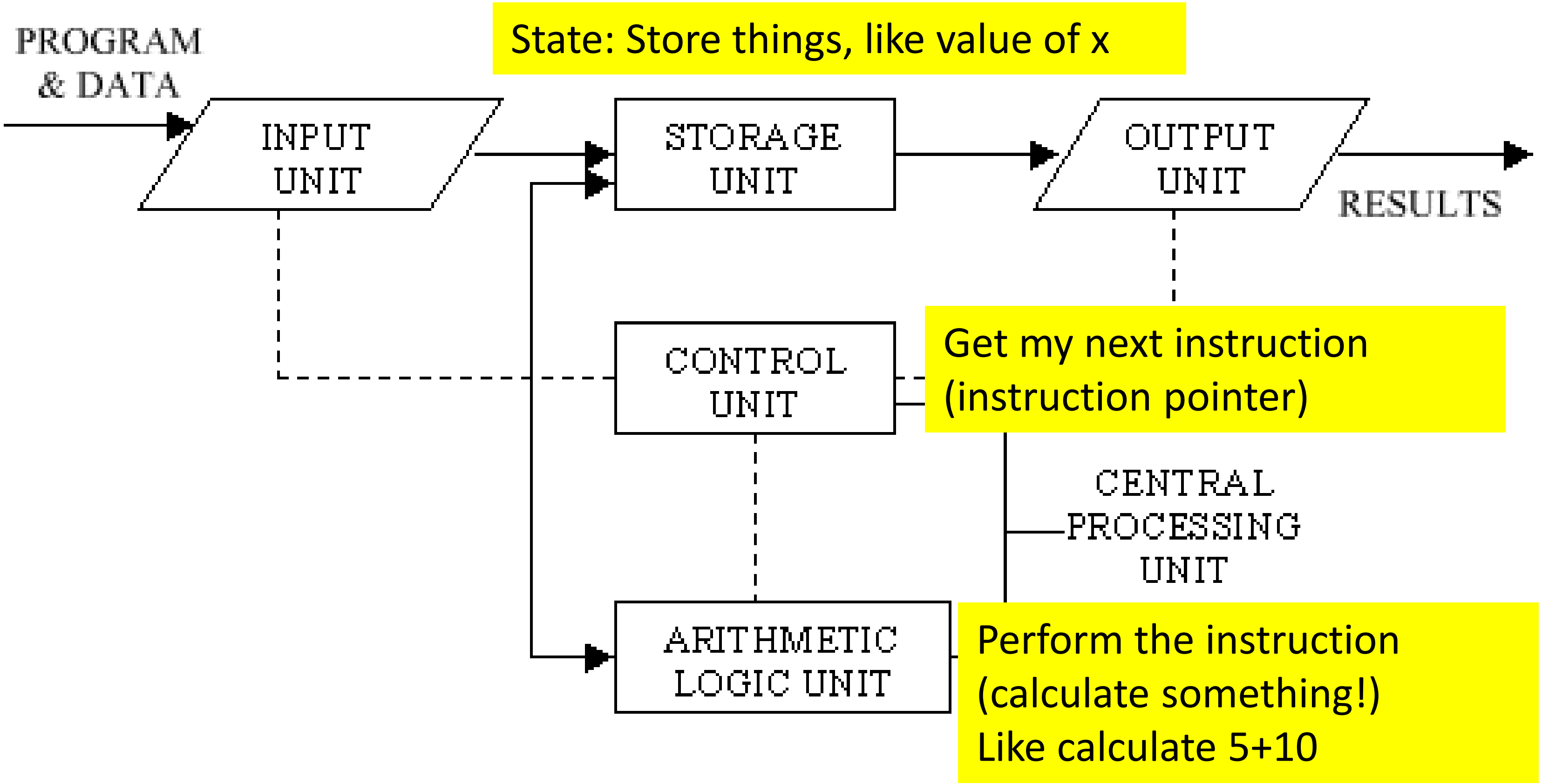
- A computer is something that predictably processes data based on a set of **instructions**
  - Like: Calculate 5+10
- Computers have an internal **state** (they can store information)
  - Like: Store the answer in a variable called x
- All computer programs can be reduced to this simple process
- Abstract mathematical idea:

  Turing Machine (1936)

What is a computer?
Hardware Simplified



What is a computer?
Hardware Simplified



More interesting program /home/2G03/mi/moreinteresting.cpp

```
#include <iostream>

int main() {
    int x;
    x = 5;
    x = x+10;
    std::cout << “x currently equals ” << x <<“\n”;
}
```

Has a main function, can be compiled into a runnable program.

How does it actually work? /home/2G03/mi/moreinteresting.cpp

```
#include <iostream>


int main() {
   int x;
   x = 5;
   x = x+10;
   std::cout << “x currently equals ” << x <<“\n”;
}
```

How does it work? /home/2G03/mi/moreinteresting.cpp

```
#include <iostream>

int main() {
   int x;
   x = 5;
   x = x+10;
   std::cout << “x currently equals ” << x <<“\n”;
}
```

Up to main, the OS does a lot behind the scenes – after main its all <u>your</u> code

/home/2G03/mi/moreinteresting.cpp

```
#include <iostream>

int main()
{
   int x;
   x = 5;
   x = x+10;
   std::cout << “x currently equals ” << x <<“\n”;
}
```

Opportunity to see how a computer Works

- Instruction pointer
- State (stored information in memory)

Added in more prints /home/2G03/mi/mii.cpp

```
1 #include <iostream>
2
3 int main()
4 {
5    int x;
     std::cout << “x=“  << x << “ after line 5\n”;
6    x = 5;
     std::cout << “x=“ << x << “ after line 6\n”;
7    x = x+10;
     std::cout << “x=“ << x << “ after line 7\n”;
8    std::cout << “x currently equals ” << x <<“\n”;
     std::cout << “x=“ << x << “ after line 8\n”;
9 }
```

Opportunity to see how a computer Works

- Instruction pointer
- State (stored information in memory)

/home/2G03/mi/mii.cpp

```
1 #include <iostream>
2
3 int main()
4 {
5    int x;
     std::cout << “x=“  << x << “ after line 5\

6    x = 5;
     std::cout << “x=“ << x << “ after line 6\
7    x = x+10;
     std::cout << “x=“ << x << “ after line 7\
8    std::cout << “x currently equals ” << x <<“\n”;
     std::cout << “x=“ << x << “ after line 8\n”;
9 }
```

Opportunity to see how a computer Works

- **Instruction pointer**
  Programs execute instructions in order
- **State** (stored information in memory)
  You can look at memory (variables). It doesn’t change unless you change it

```
[wadsley@phys-ugrad ~/mi]$ c++ mii.cpp -o mii
[wadsley@phys-ugrad ~/mi]$ mii
x=0 after line 5
x=5 after line 6
x=15 after line 7
x currently equals 15
x=15 after line 8
[wadsley@phys-ugrad ~/mi]$
```

Debugging

- Successful compiling is only half the battle
- The program must also produce correct output
- Print debugging is one strategy (see previous slide)
- Tools such as **debuggers** allow you to run code and stop at any point to query what is going on (more later)

Key to all programming

- Computers do what you tell them
- They do it in the order you say (*)
- You can always ASK the program what it currently thinks is going on
  - Just print the variable, like x

```
std::cout << x;
```

- If your program doesn't work – just go step by step and find out where it goes wrong
  - Print as much as you need to

Programming

Programming is converting your instructions into a high-level language. Once you can program in one language, it is just a matter of translation. You can even use multiple languages for a single program if necessary.

The key to programming is knowing what set of instructions will solve the problem at hand.

Pseudo Code

- Since most programming languages have the same features, it is common to write out detailed steps in a language independent way
- This is often called **pseudo code**

e.g.

pseudocode	a+b → c	Calculate a+b and store in c
C++:	c=a+b;	Math in particular looks very similar in most programming languages

Note: in programming the symbol = means assignment, it does not mean test for equality. It always means: the thing on the left is assigned the value of the expression on the right

Programming Languages

Programming Languages, e.g. C, C++, Fortran

- Built on machine code
- Contain same common features

Most of the differences between languages are just different names for the same features

Programming Languages: Common Features

1. Variables
2. Math operations
3. Tests: Logical operations
4. *Branching: Ways to choose what to do next
5. *Loops: Ways to repeat code
6. I/O: Ways to print results or read in data
7. *Functions: Ways to run code elsewhere and get information back

* Here you explicitly tell the computer not to just do the next instruction, but to continue somewhere else

Common Features: Variables

Combinations of letters indicate stored values, just like algebraic symbols

- Integers $n=2$ $j=5$ $k=-1$
- Real Numbers $x=2.67$ $mass=3.55x10^{-10}$
- Strings of characters greet = "hello"
- Logical results answer = true
- Indexed arrays of values $a_1=1$ $a_2=4$

Common Features: Math Operations

Straight from algebra

Except: In computing x=y implies x takes the value of y and y is unchanged

Thus all operators like +,-,… must be on the right hand side: x = 1+y (GOOD) x-1 = y (BAD)

- Integers n = 4/5 = 0

  Note: Integer division j = 6/5 = 1
- Real Numbers x = 2.67+3.55
- Real division y = 6./5. = 1.2
- Multiply * Divide / Add + Subtract -
- Math functions cos, sin, sqrt, …

Common Features:
Tests

Tests ask a question and get a result of true or false

- Does a equal b?
- Is “apple” before “aardvark” in the phonebook?

Often its permissible to store the result in a logical variable:

- answer = (Is a less than b?)

Common Features: Branches

- Every language has branches
- Branching moves to a different part of the program and continues from there

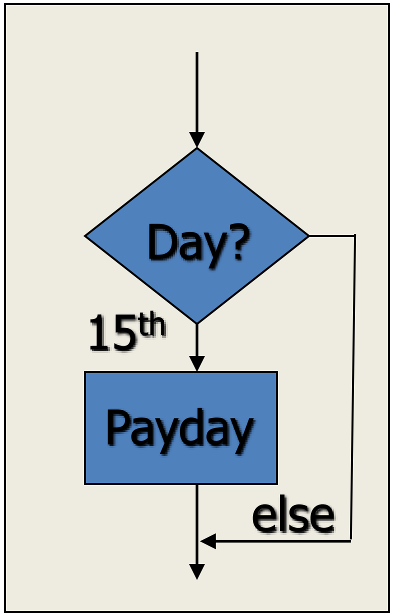
e.g.

…

if (day equals 15th of the Month) then

  Pay Employees

…



Pseudo Code

This is a flow chart

It shows where the program control goes

Common Features: Loops

Loops are a means to repeat code, often with a slightly changed value of a variable

e.g.

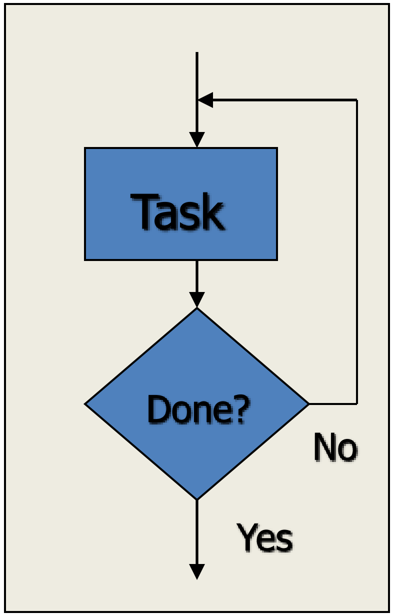
…

for each student in class

  calculate grade

…

Pseudo Code



This is a flow chart

It shows where the program control goes

Common Features: Loops

Loops are a means to repeat code, often with a slightly changed value of a variable

e.g.

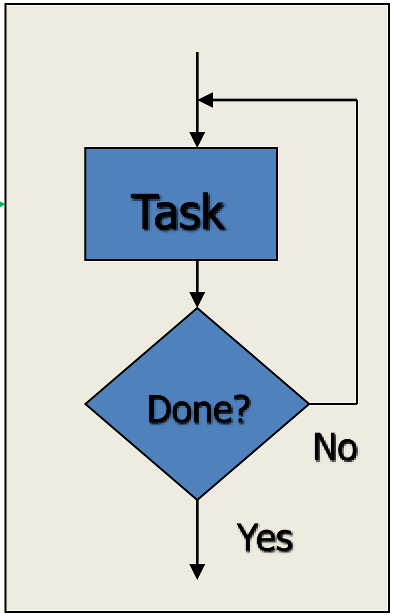
…

for each student in class

  calculate grade

…

Only valuable if a new useful task is done each time, e.g.
If the student is different each time to calculate the grade



Pseudo Code

This is a flow chart
It shows where the program control goes

Common Features:
I/O: Input and Output

- We can get values from the user and/or from a file with data into variables
- We can print out values from variables to the screen or files as needed

Common Features: Functions

Every language has a way to **call** code outside the current code and then come back

- Outside code is called a **function (**also: **method, procedure, subroutine…)**

The key things to know are:

- What data does it need? (arguments)
- What data does it change or send back? (return values)