The Basics of a C++ Program

(slides will be posted on Blackboard)

Anatomy of a C++ Program

General structure / overview of a C++ program:

```
preprocessor directives

using directive(s);


int main() {
    variable declarations;
    statements;
    return 0;
}
```

Anatomy of a C++ Program

The "Volume of a Box"

```
1  //
2  //  This program computes the volume of a box
3  //
4  #include <cstdlib>
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     /*  Declare and initialize objects */
11     double length(20.75), width(11.5);
12     double height = 9.5;
13     double volume;
14
15     /*  Calculate volume as length times width times height. */
16     volume = length * width * height;
17
18     /*  Print the volume. */
19     cout << "The volume is  " <<  volume ;
20     cout << " units cubed." << endl;
21
22     system("PAUSE");
23     //  Exit program.
24     return 0;
25 }
26
```

Anatomy of a C++ Program

Preprocessor directives

```
1  //
2  //  This program computes the volume of a box
3  //
4  #include <cstdlib>
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     /*  Declare and initialize objects */
11     double length(20.75), width(11.5);
12     double height = 9.5;
13     double volume;
14
15     /*  Calculate volume as length times width times height. */
16     volume = length * width * height;
17
18     /*  Print the volume. */
19     cout << "The volume is  " <<  volume ;
20     cout << " units cubed." << endl;
21
22     system("PAUSE");
23     //  Exit program.
24     return 0;
25 }
26
```

Anatomy of a C++ Program

The "using" directive

```
1  //
2  //  This program computes the volume of a box
3  //
4  #include <cstdlib>
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     /*  Declare and initialize objects */
11     double length(20.75), width(11.5);
12     double height = 9.5;
13     double volume;
14
15     /*  Calculate volume as length times width times height. */
16     volume = length * width * height;
17
18     /*  Print the volume. */
19     cout << "The volume is  " <<  volume ;
20     cout << " units cubed." << endl;
21
22     system("PAUSE");
23     //  Exit program.
24     return 0;
25 }
26
```

Anatomy of a C++ Program

The main() function

```
1  //
2  //  This program computes the volume of a box
3  //
4  #include <cstdlib>
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     /*  Declare and initialize objects */
11     double length(20.75), width(11.5);
12     double height = 9.5;
13     double volume;
14
15     /*  Calculate volume as length times width times height. */
16     volume = length * width * height;
17
18     /*  Print the volume. */
19     cout << "The volume is  " <<  volume ;
20     cout << " units cubed." << endl;
21
22     system("PAUSE");
23     //  Exit program.
24     return 0;
25 }
26
```

Anatomy of a C++ Program

Variable declarations

```
1  //
2  //  This program computes the volume of a box
3  //
4  #include <cstdlib>
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     /*  Declare and initialize objects */
11     double length(20.75), width(11.5);
12     double height = 9.5;
13     double volume;
14
15     /*  Calculate volume as length times width times height. */
16     volume = length * width * height;
17
18     /*  Print the volume. */
19     cout << "The volume is  " <<  volume ;
20     cout << " units cubed." << endl;
21
22     system("PAUSE");
23     //  Exit program.
24     return 0;
25 }
26
```

Anatomy of a C++ Program

Statements

```
1  //
2  //  This program computes the volume of a box
3  //
4  #include <cstdlib>
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     /*  Declare and initialize objects */
11     double length(20.75), width(11.5);
12     double height = 9.5;
13     double volume;
14
15     /*  Calculate volume as length times width times height. */
16     volume = length * width * height;
17
18     /*  Print the volume. */
19     cout << "The volume is  " <<  volume ;
20     cout << " units cubed." << endl;
21
22     system("PAUSE");
23     //  Exit program.
24     return 0;
25 }
26
```

Anatomy of a C++ Program

Comments

```
1  //
2  //  This program computes the volume of a box
3  //
4  #include <cstdlib>
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     /*  Declare and initialize objects */
11     double length(20.75), width(11.5);
12     double height = 9.5;
13     double volume;
14
15     /*  Calculate volume as length times width times height. */
16     volume = length * width * height;
17
18     /*  Print the volume. */
19     cout << "The volume is  " <<  volume ;
20     cout << " units cubed." << endl;
21
22     system("PAUSE");
23     //  Exit program.
24     return 0;
25 }
26
```

Comments

C++ has them in two flavors:

```
// a comment

/* also a comment */
```

Important distinctions:

- the first type of comment (`// a comment`) extends from the two slashes to the end of the line; it is commonly called a single-line comment
- the second type (`/* a comment */`) encompasses everything between the start and end delimiters, so it can span multiple lines of code.

Comments

Here's an example of a multi-line comment:

```
/* This is a multiple-line comment.

   My statement of the obvious is masterful.

   Of course it is! So, I demand... a SHRUBBERY!

   ...one that looks nice and is not *too* expensive.

*/
```

And here are multiple single-line comments (yay?):

```
// Here are multiple single line comments

// occurring one after the next

// giving the appearance of multiple-line comments

// I know, I know... This blows your mind. Sorry about that.
```

Comments - why use them?

Comments are for the programmer's benefit!

- comments are completely ignored by the compiler
- they provide documentation for what the code is doing (or trying to do)
- consider them a necessity if your code will be read by others
- you'll lose points if you don't comment, and the final project requires them!

Comments make your code much more readable!

Tips:

- don't wait until you're finished with your code to write comments; consider writing the comment before you write your code
- be as concise as possible while still being meaningful; write about your intent for the code, not necessarily about how your code is working.

Comments

How much commentary is enough?

- all programs you write should have an opening comment describing the purpose of your application
- all equations you use should be defined in your comments--don't just C&P a URL!
- beyond that, comment as much as you feel necessary to describe the flow and logic of your program
- ideally, someone should be able to recreate a program very similar to yours by reading only your comments (ignoring your code).
- Don't clutter your code with unhelpful commentary like this:

```
// defining variable x

double x = 3; // setting x = 3
```

- consider your audience proficient in C++; they will be able to read your code--not just your comments!

Comments

Comments can appear anywhere:

- on a line above code:

```
// gives PI to 15 decimal places
double PI = acos(-1);
```

- after code on the same line:

```
double PI = acos(-1); // gives PI to 15 decimal places
```

- between code (uncommon):

```
double PI = /* this is okay */ acos(-1);
```

- also between code (*very* uncommon ^_^):

```
double PI = // this is not... acos(-1);
```

Develop a consistent style of your liking and then stick with it...

- consistency in your code makes it much more readable (and we like readable)

“Proper” Indenting

My brain hurts. =(

```
1 #include <cstdlib>
2 #include <iostream>
3 using namespace std;
4 int main() {
5 double length(20.75), width(11.5);double height = 9.5; double volume;
6 volume = length * width * height; cout <<
7 "The volume is  " <<  volume ; cout << " units cubed." << endl;
8 system("PAUSE"); return 0;}
9
```

“Proper” Indenting

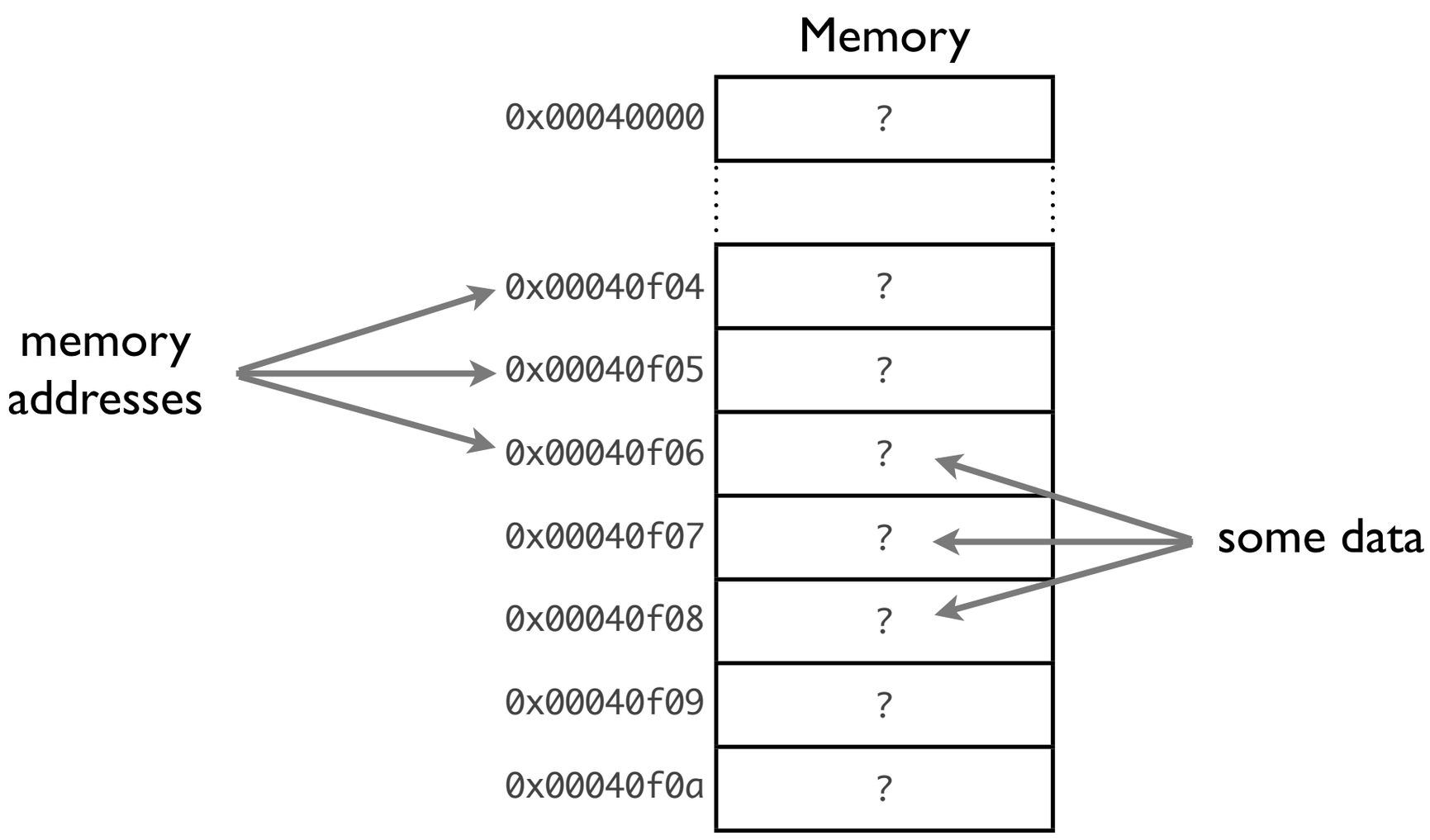
Ah… Much better!

```
1  //
2  //  This program computes the volume of a box
3  //
4  #include <cstdlib>
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     /*  Declare and initialize objects */
11     double length(20.75), width(11.5);
12     double height = 9.5;
13     double volume;
14
15     /*  Calculate volume as length times width times height. */
16     volume = length * width * height;
17
18     /*  Print the volume. */
19     cout << "The volume is  " <<  volume ;
20     cout << " units cubed." << endl;
21
22     system("PAUSE");
23     //  Exit program.
24     return 0;
25 }
26
```

Variables

Computers manipulate data stored in memory

- memory is divided into byte-sized pieces, each of which can store a single value
- each ‘slot’ has its own unique address that is used to reference it



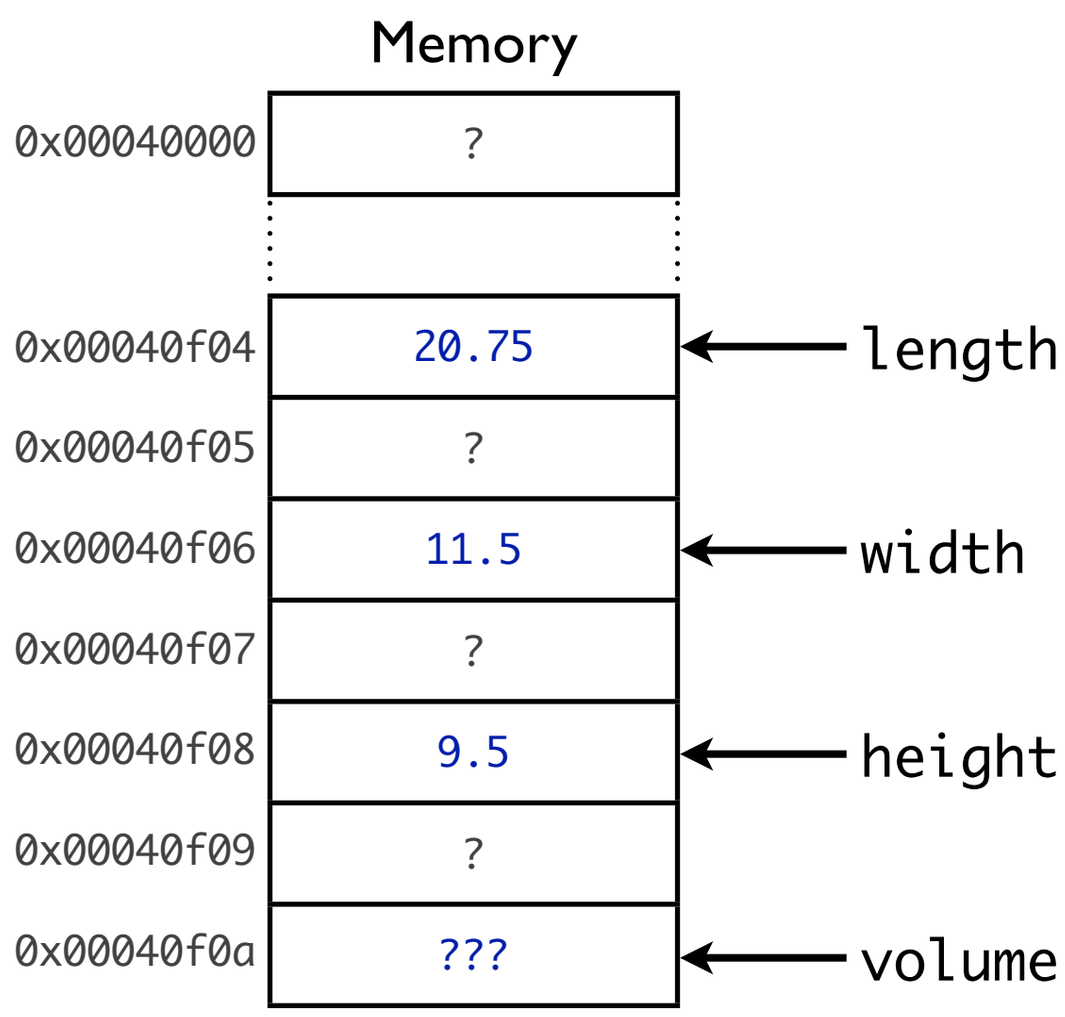
Variables

Variables provide an easy way to reference memory locations:

```
double length(20.75), width(11.5);

double height = 9.5;

double volume; // some garbage value
```



Say that again?

Variables

Computers manipulate data stored in memory

- think of memory like a city of data
- every block (byte) of memory has its own address, just like every house in a real city has a street address
- we can access data in memory by going to a specific address...
- or, we can give a name to the memory address and use it instead (much nicer!)

A variable is a named chunk of memory

- variables basically store data
- we can access or modify the data just by using the variable's name

Variables store data.

Simple, right? =)

Variables

Variables have the following properties:

- a name, or *identifier*
- a data *value*
- a data *type*
- a data *size* (corresponds to data type)
- a *location* in memory

We mostly care about the *name*, *type*, and *value* of a variable

- the other properties are nice and all, but meh...

Variables

Rules for naming variables:

- identifiers can only contain letters, numbers, and the underscore character (_)
- identifiers cannot start with a number
- an identifier must not be a C++ keyword (see table 2.1 on page 41 in your book)
- identifiers are case-sensitive (`myvar` is different than `myVar` and `MYVAR`)

These rules apply to all *identifiers* in C++, not just variable names!

Valid or Invalid?

`grade%`

INVALID

Identifiers can only contain letters, numbers, and the underscore character

Valid or Invalid?

```
x_sum
```

VALID

Valid or Invalid?

```
1st_item
```

INVALID

Identifiers cannot begin with numbers

Valid or Invalid?

```
item1
```

VALID

Valid or Invalid?

```
first item
```

INVALID

Identifiers cannot contain spaces
(on their own, these are two valid identifiers)

Valid or Invalid?

```
first-item
```

INVALID

Identifiers cannot contain dashes
(this is actually trying to do subtraction)

Valid or Invalid?

new

INVALID

Identifiers cannot be C++ keywords
(google or check your book if unsure)

Valid or Invalid?

`myFriend`

VALID

Valid or Invalid?

`friend`

INVALID

Identifiers cannot be C++ keywords
(google or check your book)

Valid or Invalid?

```
main
```

VALID

Try it and see. =)

Variables

Examples of declaring variables:

```
// a double variable called number with value 2.5

double number = 2.5;

// same as above, but using a different syntax

double number(2.5);

// a double variable called volume

double volume;
```

You can also chain variable declarations using commas:

```
// three double variables

double num1 = 1, num2(2), num3;
```

Data Types

Description	Keyword	Examples
integers	`int`	`1, 2, 0, -3`
real numbers	`double`	`3.14, 0.001, 10.5`
boolean true/false	`bool`	`true, false`
single characters	`char`	`'a', '1', '/'`
strings of text	`string`	`"Trespassers will be shot; survivors will be shot again"`

Notice that:

- strings are delimited by *double* quotes
- chars are delimited by *single* quotes

Uninitialized string variables get the value of an empty string (""), not a garbage value!

Constants

Constants are simply variables whose value can't be changed once set

Syntax: just add `const`:

```
const double PI = 3.14159;
```

Use symbolic constants for:

- numbers such as π and physical constants (gravitational constant, for example)
- variables whose value you know won't change during your program, such as the width of a column in a table
- values that someone running your program might want to change (makes it easier)
- any static data that is reused in your code

Assignment Operator (=)

The assignment operator (=) assigns a new value to a variable:

```
double height = 10; // declare 'height' and assign it the value 10
height = 9.5;       // assign 'height' a new value, 9.5
```

Read it like "height is assigned the value 9.5"

- But why, when "equals" is so much more terse?
- Meet the equality operator (==) with its double equal sign combo! Bwahaha!
- Your life just got more difficult. I should be sorry for you, but I'm not.

Simple Arithmetic Operators

+	addition operator
-	subtraction operator
*	multiplication operator
/	division operator
%	modulus (remainder) operator

Anatomy of a C++ Program

The "Volume of a Box" again…

```
1  //
2  //  This program computes the volume of a box
3  //
4  #include <cstdlib>
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     /*  Declare and initialize objects */
11     double length(20.75), width(11.5);
12     double height = 9.5;
13     double volume;
14
15     /*  Calculate volume as length times width times height. */
16     volume = length * width * height;
17
18     /*  Print the volume. */
19     cout << "The volume is  " <<  volume ;
20     cout << " units cubed." << endl;
21
22     system("PAUSE");
23     //  Exit program.
24     return 0;
25 }
26
```

Generating Output

For now, output is what we see in the black window that pops up.

We write to that little black window with `cout` and `<<`:

- `cout` is a predefined stream variable
- `<<` is the output operator

Examples:

```
cout << 10;       // outputs the number 10
cout << “hello”; // outputs the word “hello” (a string)


double num = 3013;
cout << num;      // outputs the value stored in num (a double variable)
```

Generating Output

You can use endl to move to the next line:

```
// outputs the number 10 followed by a new line
cout << 10 << endl;


// outputs the word “hello” (a string) followed by a new line
cout << “hello” << endl;
```

```
// outputs the value stored in num (a double variable)
double num = 3013;
cout << num << endl;
```

Generating Output

You can chain multiple output statements together:

```
// 1020
cout << 10 << 20 << endl;


// 10 20
cout << 10 << “ ” << 20 << endl;


// 10
// 20
cout << 10 << endl << 20 << endl;


// My favorite number is 10
cout << “My favorite number is ” << 10 << endl;
```

Generating Output

Recognize this?

```
/* Print the volume. */

cout << “The volume is ” << volume;

cout << “ units cubed.” << endl;
```

It’s pretty easy, and you’ll be a PRO before the end of the semester

Try it.

Getting Input from the User

We get input from that little black window with `cin` and >>:

- `cin` is a predefined stream variable
- >> is the input operator (>> for input; << for output)

Example (includes an appropriate prompt):

```
int num;                          // declare a variable to store the value
cout << “Enter an integer: ”;     // prompt the user for a number
cin >> num;                       // read a value from the user
```

You can display the new value to convince yourself it worked:

```
cout << “You entered: ” << num;
```

Getting Input from the User

Recipe:

- declare a variable of the appropriate type to store the input
- display (output) an appropriate prompt to the user. Let him know what he needs to enter.
- use `cin` to read a value into the previously declared variable.

Another example:

```
string name;

cout << “Please enter your first name: ”;

cin >> name;

cout << “Hello, ” << name << “!” << endl;
```

Try it.