CSCI 262
Data Structures

2 - Review

Welcome Back

What you learned in CSCI 261 (or equivalent):
- Variables
- Types
- Arrays
- Expressions
- Conditionals
- Branches & Loops
- Functions
- Recursion
- Classes & Objects
- Streams
- Vectors
- Strings

You remember all of this, right?

Hello, Let's Review

Here's a simple C++ program:

```cpp
#include <iostream>
using namespace std;

int main() {
    string hello = "Hello, world!";
    cout << hello << endl;
    return 0;
}
```

This tells the compiler that it should include symbols and types from the standard library `iostream`.

This is boilerplate that lets you use standard library symbols without extra ugly syntax. Don’t worry about it for now, just put it in whenever you #include something.

Hello, Let's Review

How to Review

- Remaining slides:
  - Some new material – e.g., function overloading
  - Mostly review

- Your responsibility:
  - Go through all the slides that follow
  - Note any questions on old or new concepts
  - Try to learn concept from textbook
  - Ask instructor if you still have questions!
**Variables**

Declaration:

```c
int x;
```

Use in expressions:

```c
x + 10
```

Set via assignment operator:

```c
x = 4;
```

Declare and initialize:

```c
int x = 42;
```

---

**Types**

- Basic types
  - Integer types:
    - `int`: 42, -99, 103482039
    - `unsigned`: like `int`, but no negative values
  - `char`: ‘k’
  - Floating point types:
    - `double`: 3.14159, 4.5e3, -0.0001
  - Boolean type:
    - `bool`: true, false
- Pointers
- Arrays
- Class/struct types

---

**Expressions & Operators**

Working definition: anything with a value is an expression:

- Variables
  - `x`
  - Indexed array variables
  - `arr[10]`
- Literals
  - `42`
  - `"Hello"`
  - `true`
- Function calls returning a value
  - `sqrt(17)`
- Arithmetic/logical expressions using operators:
  - `4 + 7 / 3.0`
  - `(x * sqrt(2)) + 1) % y`
  - `count == 0`
  - `a || b && c`

---

**Loops**

What if we want to print “Hello, world!” three times?

```c
for (int i = 1; i <= 3; i++) {
    cout "Hello, world!" << endl;
}
```

Output:

1 Hello, world!
2 Hello, world!
3 Hello, world!

---

**Another Loop**

```c
int i = 3;
while (i > 0) {
    cout "Hello, world!" << endl;
    i--;
}
```

Output:

3 Hello, world!
2 Hello, world!
1 Hello, world!

---

**Conditionals**

```c
if (true-false-expression) {
    true-block
} else {
    false-block
}
```

Also should know use of: break continue
Hello, if?

Let’s modify Hello to respond to an input:

```
... char answer;
    cout << "Say (H)ello or (G)oodbye?" << endl;
    cin >> answer;
    if (answer == 'H') {
        cout << "Hello, world!" << endl;
    } else {
        cout << "Goodbye, world!" << endl;
    }
... What happens if the user enters "h" instead of "H"?
```

Arrays

```
... int numbers[3];
    numbers[0] = 14;
    numbers[1] = -3;
    numbers[2] = 7093;
...
```

Oops! What’s wrong here?

Arrays

```
... int numbers[3];
    numbers[0] = 14;
    numbers[1] = -3;
    numbers[2] = 7093;
...
```

Let’s print out the numbers in the array.
What about in reverse order?

Loops on Arrays

```
... int numbers[] = {14, -3, 7093};
    for (int i = 0; i < 3; i++) {
        cout << numbers[i] << endl;
    }
    for (int i = 2; i >= 0; i--) {
        cout << numbers[i] << endl;
    }
```

Array initialization – only when array is declared!

Functions

```
We’ve seen one function:
    int main() { ... }

Here’s another:
    int print_it(string msg) {
        cout << msg << endl;
        return msg.length();
    }
```

A member function or method of the string class.
Hello Functions!

A silly program.

```c++
#include <iostream>
#include <string>
#include <cmath>
using namespace std;

int print_it(string msg) {
    cout << msg << endl;
    return msg.length();
}

int main() {
    int n;
    double root;
    n = print_it("Hello, world!");
    root = sqrt(n);
    cout << "The square root of the number of characters printed is ";
    cout << root << endl;
    return 0;
}
```

Note, we have to declare a function before we use it. (Could also use a prototype.)

Recursion

Functions can call themselves.

```c++
void print_n_times(string s, int n) {
    if (n == 0) return;
    cout << s << endl;
    print_n_times(s, n - 1);
}
```

Base case. Very important!

Recursive call. Note that the parameter `n` moves towards the base case condition.

Function Overloading

- C++ allows multiple functions of the same name:
  ```c++
  void print_it(int x) {
    cout << "an integer: " << x << endl;
  }
  
  void print_it(string s) {
    cout << "a string: " << s << endl;
  }
  
  What to call based on the parameter list
  - So parameter lists must be different for each overload
  - Can get confusing when mixed with type promotion:
    ```c++
    print_it(3.1415); // what does this do?
    ```
  ```

Default Parameters

Alternative when one overload is just a specialized version of another:

```c++
// prints n times, or just once if n omitted
void print_n_times(string s, int n = 1) {
    for (int j = 0; j < n; j++) {
        cout << s << endl;
    }
}
```

With the above, we can do:

```c++
print_n_times("Hello", 10); // prints Hello 10 times
print_n_times("Goodbye"); // prints Goodbye once
```

Rules:
- Cannot omit earlier parameters, supply later ones
- Cannot overload if parameter list is interpretable as call to function with default params omitted, e.g., cannot also define:
  ```c++
  void print_n_times(string s) {
  ```

Parameter by Value or Reference

What does this program print?

```c++
void set_to_zero(int x) {
    x = 0;
}

int main() {
    int n = 42;
    set_to_zero(n);
    cout << n << endl;
}
```

Answer: 42
Parameter passed by value

Passing Parameters by Reference

```c++
void set_to_zero(int &x) {
    x = 0;
}

int main() {
    int n = 42;
    set_to_zero(n);
    cout << n << endl;
}
```

This prints: 0
The Stack

- Holds “stack frames” aka “activation records”
- Each function call results in a new stack frame
- Each stack frame contains memory for:
  – Local variables declared in the function
  – Arguments passed into function
  – Return address for function
- When the function is exited, all of this memory is returned to the stack automatically.

Function Call Example

```c++
void quotient(double num, double den) {
    double q = num / den;
    cout << num << '/' << den << ' ' "is " q " endl; |
}
void print_quotients(int x, int y) {
    quotient(x, y);
    quotient(y, x);
}
int main() {
    int a, b;
    cout << "Please enter 2 non-zero integers: ";
    cin >> a >> b;
    print_quotients(a, b);
    return 0;
}
```

Example

At start of main():
```c++
int main() {
    int a, b;
    cout << "Please enter 2 non-zero integers: ";
    cin >> a >> b;
    print_quotients(a, b);
    return 0;
}
```

Example

After getting input:
```c++
main  Stack

int a = 7
int b = 2
return address

Top of Stack

cout << "Please enter 2 non-zero integers: ";
cin >> a >> b;
print_quotients(a, b);
return 0;
```

Example

At beginning of call to print_quotients:
```c++
void print_quotients(int x, int y) { |
    quotient(x, y);
}
```

Example

At beginning of first call to quotient:
```c++
void quotient(double num, double den) { |
    double q = num / den;
    cout << num << '/' << den << ' ' "is " q " endl; |
}
```
Example

At end of call to quotient:

```c
> Please enter 2 non-zero integers: 7 2
> 7/2 is 3.5
```

```c
top
print_quotients
main
int a = 7
int b = 2
return address
quotient
double num = 7
double den = 2
double q = 3.5
return address
Top of Stack
```

Example

After return from call to quotient:

```c
> Please enter 2 non-zero integers: 7 2
> 7/2 is 3.5
```

```c
main
int a = 7
int b = 2
return address
print_quotients
int x = 7
int y = 2
return address
quotient
double num = 7
double den = 2
double q = 3.5
return address
Top of Stack
```

Example

At beginning of second call to quotient:

```c
> Please enter 2 non-zero integers: 7 2
> 7/2 is 3.5
```

```c
main
int a = 7
int b = 2
return address
print_quotients
int x = 7
int y = 2
return address
quotient
double num = 2
double den = 7
double q = ?
return address
Top of Stack
```

Example

At end of second call to quotient:

```c
> Please enter 2 non-zero integers: 7 2
> 7/2 is 3.5
> 2/7 is 0.285714
```

```c
main
int a = 7
int b = 2
return address
print_quotients
int x = 7
int y = 2
return address
quotient
double num = 2
double den = 7
double q = 0.285714
return address
Top of Stack
```

Example

After return from second call to quotient:

```c
> Please enter 2 non-zero integers: 7 2
> 7/2 is 3.5
> 2/7 is 0.285714
```

```c
main
int a = 7
int b = 2
return address
print_quotients
int x = 7
int y = 2
return address
Top of Stack
```

Example

After call to print_quotients:

```c
> Please enter 2 non-zero integers: 7 2
> 7/2 is 3.5
> 2/7 is 0.285714
```
CLASSES AND OBJECTS

CLASSES

- Objects also have type. Objects of the same type:
  - Have a common set of properties and methods
  - Used in a similar manner to primitive types.
- Types are (usually) modeled by classes. Classes formally define the properties and methods.
- Essentially, defining classes is a way to add new types to C++.
  (Classes do some other neat things, too, but we’ll get to that later.)

Classes in C++

Classes are created via a class declaration:

```cpp
class student {
  public:
    string name;
    string year;
    double gpa;
    bool is_hungry;

    student();
    void eat();
    void sleep();
    void program(int assignment);
};
```

Defining Member Functions

The declaration only gave the member function signatures (prototypes); we still have to write the functions themselves:

```cpp
void student::eat() {
  is_hungry = false;
}

void student::program(int assignment) {
  if (grade(this, assignment) == 'A') gpa++;
}
```

Using Objects in C++

- Objects can be created just like chars, ints, etc.:
  ```cpp
  student s;
  ```
- Properties are referenced by the “.” operator:
  ```cpp
  s.name = "April";
  s.gpa = 4.0;
  double d = s.gpa;
  ```
- Methods are invoked on objects also using “.”:
  ```cpp
  s.sleep();
  ```

Objects

C++ is an object-oriented (OO) language.

What’s an object?

A working definition: An object is a package of data with associated behavior.

More specifically, we say that an object has properties (fields, attributes, data, state), and that it has associated methods (functions).
Some Notes on Visibility

- Many philosophies around visibility
  - “All data should be private”
  - Partly a matter of style

- Rule of thumb:
  - If it is specific to the implementation, it is private
  - Else, it is public

- Not all OO languages have visibility modifiers. (But they all have commenting systems!)

Streams

- Console I/O:
  ```
  #include <iostream>
  cin >> some_var;
  cout << expression << endl;
  string s;
  getline(cin, s); // must #include <string>
  ```

- File I/O:
  ```
  #include <fstream>
  ifstream fin("words.txt");
  fin >> some_var;
  getline(fin, s);
  ofstream fout("output.txt");
  fout << expression << endl;
  ```

- We’ll also learn about stringstream objects (later).

VECTORS

Arrays and Vectors

Arrays:

```cpp
int foo[10];
for (int j = 0; j < 10; j++)
    foo[j] = j;
```

Vectors:

```cpp
#include <vector>
...
vector<int> foo(10);
for (int j = 0; j < 10; j++)
    foo[j] = j; // ≈ foo.at(j) = j
```

Do More with Vectors

E.g. you can append to a vector – it automatically resizes:

```cpp
vector<int> foo;
for (int j = 0; j < 10; j++)
    foo.push_back(j);
```

foo contains:

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

And so much more: see Help page of course website for C++ documentation websites.
Something New-ish

C++ 11 added a new type of for loop:

```cpp
vector<int> numbers = {14, -3, 7093};
for (int x : numbers) {
    cout << x << endl;
}
```

Note vector initializer list - can be used almost like a literal in certain contexts.

This denotes that x is a variable of type int which will take on each value in numbers in turn.

STRINGS

About Strings

In C/C++, the literal "Hello" is called a string.
It is of type `char[]` (a char array).

Confusingly, C++ defines a new type, string.

A string is mostly interchangeable with a string (which in C++ is called a "C-string").
But, you can do more with string objects:

```cpp
#include <string>
...
string foo = "Hello"; // note assignment of string to string
string bar = "World"; // actually implicit constructor call
string hello = foo + ", " + bar + "!"; // test for equality works with string
```

More About Strings

Know/learn the string interface!

- See Help page of course website for C++ documentation websites
- Some string methods you should know:
  - `length` operator[]
  - `size` operator+
  - `find` operator+=
  - `substr` relational operators

Up Next

- Please continue to review chapters 1 – 10 in your textbook
- Friday, January 12
  - Lab 1 - Compile
  - APT 1 assigned
- Monday, January 15: MLK, Jr. Day – NO CLASS
- Wednesday, January 17
  - Abstraction
  - Lab 1 due