Whirlwind C++ review

Comp Sci 1575 Data Structures

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates

2 Tricks and tools

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates

2 Tricks and tools

Hello world


```
#include <iostream>

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

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates
2 Tricks and tools

Operating with variables


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

int main ()
{
  int a, b;
  int result;

  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  cout << result;

  return 0;
}
```

Initialization of variables


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

int main ()
{
  int a=5;                  // initial value: 5
  int b(3);                 // initial value: 3
  int c{2};                 // initial value: 2
  int result;               // undetermined

  a = a + b;
  result = a - c;
  cout << result;

  return 0;
}
```

Strings


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

int main ()
{
  string mystring;
  mystring = "Initial string content";
  cout << mystring << endl;
  mystring = "Different string content";
  cout << mystring << endl;
  return 0;
}
```

Types

Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as char. At least 8 bits.
	signed **short** *int*	Not smaller than char. At least 16 bits.
	signed **int**	Not smaller than short. At least 16 bits.
	signed **long** *int*	Not smaller than int. At least 32 bits.
	signed **long long** *int*	Not smaller than long. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than float
	long double	Precision not less than double
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

- shorthand for some in bold

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates
2 Tricks and tools

Assignment operator


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

int main ()
{
  int a, b;               // a:?,  b:?
  a = 10;                 // a:10, b:?
  b = 4;                  // a:10, b:4
  a = b;                  // a:4,  b:4
  b = 7;                  // a:4,  b:7

  cout << "a:" << a << endl;

  cout << "b:" << b << endl;
}
```

Arithmetic operators

operator	description
+	addition
-	subtraction
*	multiplication
/	division
%	modulo

Compound assignment operators


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

int main ()
{
  int a, b=3;
  a = b;
  a+=2;                    // equivalent to a=a+2
  cout << a;
}
```

Compound assignment operators

expression	equivalent to...
`y += x;`	`y = y + x;`
`x -= 5;`	`x = x - 5;`
`x /= y;`	`x = x / y;`
`price *= units + 1;`	`price = price * (units+1);`

Increment and decrement (++, –)

Example 1	Example 2

`x = 3;`
`y = ++x;`
`// x contains 4, y contains 4`
	`x = 3;`
	`y = x++;`
	`// x contains 4, y contains 3`

- Watch out for these during loops

Relational and comparison operators

operator	description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Relational and comparison operators

Suppose that a=2, b=3, and c=6, then:

```
(7 == 5)      // evaluates to false
(5 > 4)       // evaluates to true
(3 != 2)      // evaluates to true
(6 >= 6)      // evaluates to true
(5 < 5)       // evaluates to false
(a == 5)      // evaluates to false
(a*b >= c)    // evaluates to true
(b+4 > a*c)   // evaluates to false
((b=2) == a)  // evaluates to true
```

Logical operators

&& OPERATOR (and)		
a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

|| OPERATOR (or)		
a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

```
( (5 == 5) && (3 > 6) )  // evaluates to false
( (5 == 5) || (3 > 6) )  // evaluates to true
!(5 == 5)    // evaluates to false
!(6 <= 4)    // evaluates to true
!true        // evaluates to false
!false       // evaluates to true
```

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates
2 Tricks and tools

Input and output (i/o)


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

int main()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << "and double is " << i*2 << endl;
  return 0;
}
```

cin with strings


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

int main ()
{
  string mystr;
  cout << "What's your name? ";
  getline(cin, mystr);
  cout << "Hello " << mystr << endl;
  cout << "What is your favorite food?";
  getline(cin, mystr);
  cout << "You like " << mystr << endl;
  return 0;
}
```

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates

2 Tricks and tools

Conditionals


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

int main()
{
  int x;
  cout << "Please enter an integer value: ";
  cin >> x;
  if(x > 0)
    cout << "x is positive";
  else if(x < 0)
    cout << "x is negative";
  else
    cout << "x is 0";
  return 0;
}
```

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates

2 Tricks and tools

While loops


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

int main()
{
  int n = 10;

  while(n>0)
  {
    cout << n << ", ";
    --n;
  }

  cout << "liftoff! \n";
}
```

What does this do?

do while loops


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

int main ()
{
  string str;
  do
  {
    cout << "Enter text: ";
    getline (cin,str);
    cout << "You entered: " << str << '\n';
  } while (str != "goodbye");
}
```

What does this do?

for loops


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

int main()
{
  for(int n=10; n>0; n--)
  {
    cout << n << ", ";
  }
  cout << "liftoff!\n";
}
```

What does this do?

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates

2 Tricks and tools

Functions


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

int addition(int a, int b)
{
  int r;
  r = a+b;
  return r;
}

int main()
{
  int z;
  z = addition (5,3);
  cout << "The result is " << z;
}
```

void functions

have no return type

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

void printmessage()
{
  cout << "I'm a function!";
}

int main()
{
  printmessage();
}
```

Passing parameters by reference


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

void doubleVal(int &a, int &b, int &c)
{
  a*=2; b*=2; c*=2;
}

int main ()
{
  int x=1, y=3, z=7;
  doubleVal(x, y, z);
  cout << "x=" << x
  << ", y=" << y
  << ", z=" << z;
  return 0;
}
```

Default values in functions


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

int divide(int a, int b=2)
{
  int r;
  r=a/b;
  return r;
}

int main()
{
  cout << divide(12) << '\n';
  cout << divide(20,4) << '\n';
  return 0;
}
```

Defaulted parameter has to be the last one

Declaring functions prototypes part 1


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

void odd(int x);
void even(int x);

int main()
{
  int i;
  do{
    cout << "Enter a number (0 exits): ";
    cin >> i;
    odd(i);
  } while(i!=0);
  return 0;
}
.........
```

Declaring functions prototypes part 2

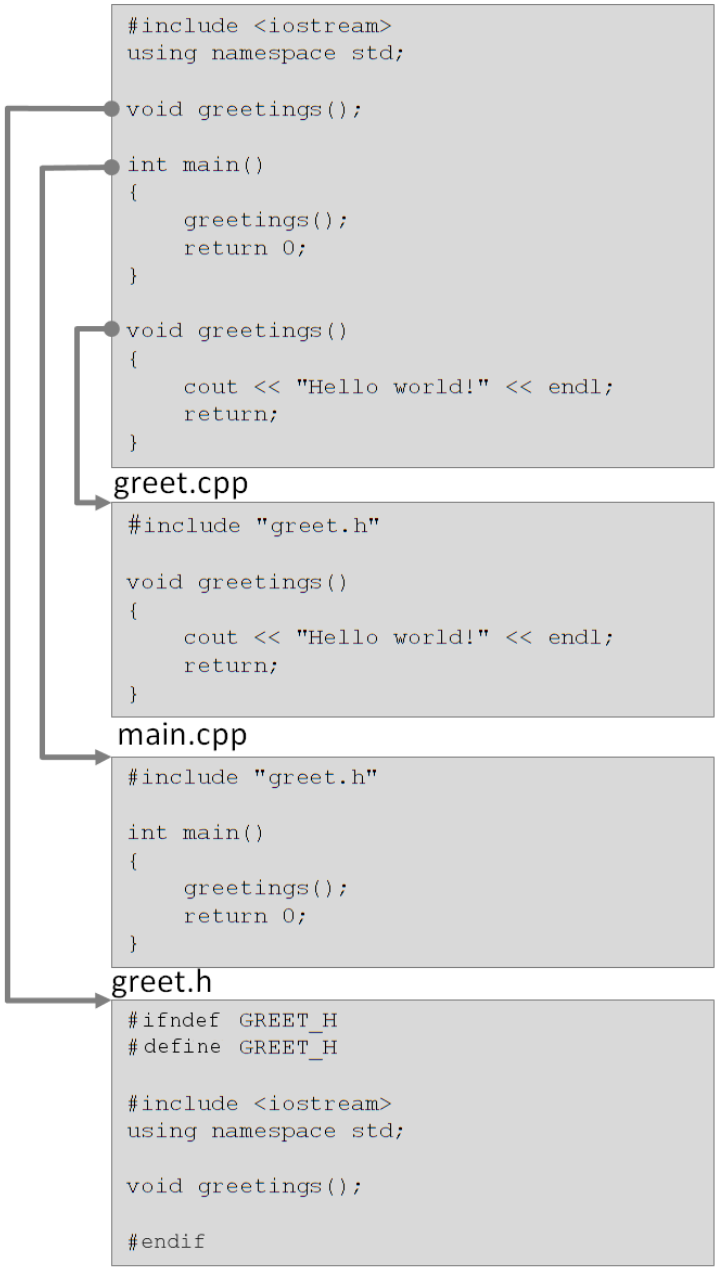

```
  ......

void odd(int x)
{
   if((x%2)!=0)
      cout << "It is odd.\n";
   else
      even (x);
}

void even(int x)
{
   if((x%2)==0)
      cout << "It is even.\n";
   else
      odd (x);
}
```

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates

2 Tricks and tools

Multiple files



Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates
2 Tricks and tools

Scope


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

int main () {
  int x = 10;
  int y = 20;
  {
    int x;   // ok, inner scope.
    x = 50;  // sets value to inner x
    y = 50;  // sets value to (outer) y
    cout << "inner block:\n";
    cout << "x: " << x << '\n';
    cout << "y: " << y << '\n';
  }
  cout << "outer block:\n";
  cout << "x: " << x << '\n';
  cout << "y: " << y << '\n';
  return 0;
}
```

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates
2 Tricks and tools

Arrays


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

int foo [] = {16, 2, 77, 40, 12071};
int n, result=0;

int main ()
{
  for ( n=0 ; n<5 ; ++n )
  {
    result += foo[n];
  }
  cout << result;
  return 0;
}
```

What does this do?

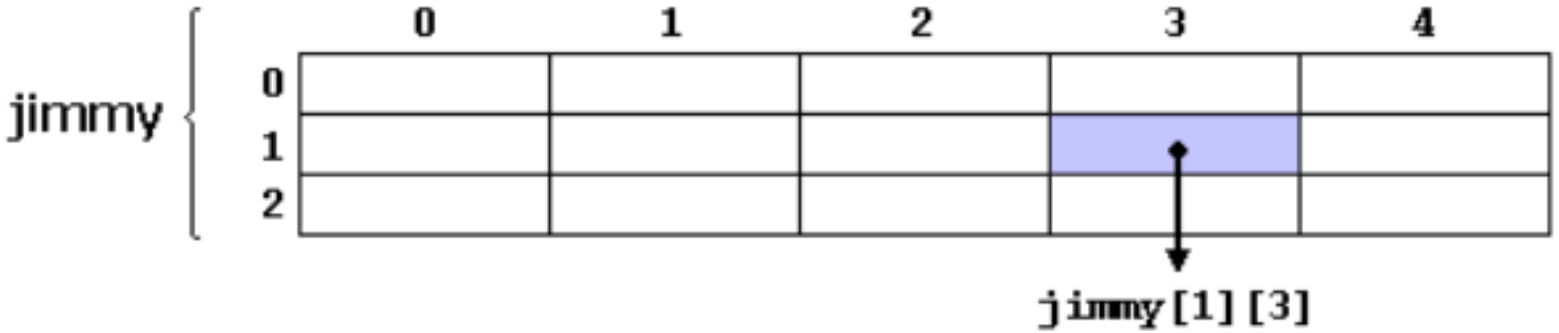
Arrays as parameters


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

void printarray(int arg[], int length) {
  for (int n=0; n<length; ++n)
    cout << arg[n] << ' ';
  cout << endl;
}

int main ()
{
  int firstarray[] = {5, 10, 15};
  int secondarray[] = {2, 4, 6, 8, 10};
  printarray(firstarray,3);
  printarray(secondarray,5);
}
```

Multidimensional arrays

Arrays are indexed by array[down][over]

jimmy[1][3]



strings and NTCS arrays


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

int main ()
{
  char question1[] = "What is your name? ";
  string question2 = "Where do you live? ";
  char answer1 [80];
  string answer2;
  cout << question1;
  cin >> answer1;
  cout << question2;
  cin >> answer2;
  cout << "Hello, " << answer1
  << " from " << answer2;
  return 0;
}
```

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates
2 Tricks and tools

Structs


```
struct type_name
{
  member_type1 member_name1;
  member_type2 member_name2;
               .
               .
               .
  member_typeN member_nameN;
};
```

Struct alternative initialization and access


```
struct type_name
{
   member_type1 member_name1;
   member_type2 member_name2;
   member_type3 member_name3;
   .
   .
   .
} object_names; // optional

// Access:
object_name.member_name;
variable = object_name.member_name;

// Assignment:
object_name.member_name = value;
```

Example struct


```
struct point
{
  float m_Xcoord;
  float m_Ycoord;
};

int main()
{
  point p1, p2;
    ...
```

Nesting structs


```
struct child_name
{
  member_type member_name;
};

struct parent_name
{
  child_name member_name;
};

int main()
{
  parent_name parent_instance;
  parent_instance.child_name.member_name = value;
    ...
```

Example struct with nested access


```
struct point
{
    float m_Xcoord;
    float m_Ycoord;
};

struct line
{
    point m_Left;
    point m_Right;
};

int main()
{
    line my_line;
    my_line.m_Left.m_Xcoord = 5;
    my_line.m_Left.m_Ycoord = 8;
```

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates
2 Tricks and tools

Classes


```
class name_of_type
{
  public:
    // -- function prototypes here

  private:
    // -- member data here
};
```

Class access specifiers


```
class class_name
{
   access_specifier_1:
     member_type member1;
   access_specifier_2:
     member_type member2;
   ...
} object_names; // optional
```

access_specifier defaults to private, with the following options:

- **private** members of a class are accessible only from within other members of the same class (or from their "friends").
- **protected** members are accessible from other members of the same class (or from their "friends"), but also from members of their derived classes.
- **public** members are accessible from anywhere where the object is visible.

Classes member access


```
class class_name
{
  access_specifier_1:
    member_type member1;
  access_specifier_2:
    member_type member2;
  ...
} object_names; // optional

int main()
{
  class_name class_instance;
  class_instance.data_element = value;
  class_instance.function();
  ...
```

Class example


```
#include <iostream>
using namespace std;
class Rectangle{
  private:
    int width, height;
  public:
    void set_values (int,int);
    int area() {return width*height;}
};
void Rectangle::set_values(int x, int y){
  width = x;
  height = y;
}
int main(){
  Rectangle rect, rectb;
  rect.set_values (3,4);
  rectb.set_values (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

Classes and uniform initialization


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

class Circle{
  private:
    double radius;
  public:
    Circle(double r){ radius = r; }
    double circum(){return 2*radius*3.14159265;}
};

int main(){
  Circle foo(10.0);   // functional form init
  Circle bar = 20.0;   // assignment init.
  Circle baz{30.0};   // uniform init.

  cout << "foo's circumference: " << foo.circum();
  return 0;
}
```

Member initialization


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

class Circle{
    double radius; // defaults to private
  public:
    Circle(double r): radius(r) { }
    double area(){return radius*radius*3.14159265;}
};

class Cylinder{
    Circle base;
    double height;
  public:
    Cylinder(double r, double h): base(r), height(h) { }
    double volume(){return base.area() * height;}
};

int main(){
  Cylinder foo (10,20);

  cout << "foo's volume: " << foo.volume() << '\n';
  return 0;
}
```

Constructors

The constructor function is declared just like a regular member function, but with a name that matches the class name and without any return type; not even void. After writing your own constructor, the default constructor will no longer work.

Constructors


```
#include <iostream>
using namespace std;
class Rectangle{
    int width, height; // defaults to private
  public:
    Rectangle(); // new default constructor
    Rectangle(int,int); // constructor
    Rectangle(const Rectangle & source); // copy constructor
    int area(void){return (width*height);}
};
Rectangle::Rectangle(){
  width = 5;
  height = 5;
}
Rectangle::Rectangle(int a, int b){
  width = a;
  height = b;
}
Rectangle::Rectangle(const Rectangle & source){
  width = source.width;
  height = source.height;
}
int main(){
  Rectangle rect(3,4);
  Rectangle rectb;
  Rectangle rectc(rect); // copy constructor
  //Rectangle rectd(); // wrong
  cout << "rect area: " << rect.area() << endl;  // 12
  cout << "rectb area: " << rectb.area() << endl; // 25
  cout << "rectb area: " << rectc.area() << endl; // 12
  return 0;
}
```

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates
2 Tricks and tools

Overloading functions


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

int sum(int a, int b){
  return a+b;
}

double sum(double a, double b){
  return a+b;
}

int main(){
  cout << sum (10, 20) << '\n';
  cout << sum (1.0, 1.5) << '\n';
  return 0;
}
```

Can be done in your own types and classes

Overloading operators

type operator sign (parameters) { /*... body ...*/ }

```
Can overload operators too:
+   -   *   /   =   <   >   +=   -=   *=   /=   <<   >>
<<=  >>=  ==   !=   <=   >=   ++   --   %   &   ^   !   |
~   &=   ^=   |=   &&   ||   %=   []   ()   ,   ->*   ->
new   delete   new[]   delete[]
```

Some operators may be overloaded in two forms:

1. as a member function:
   Parameter expected for a member function overload for operations such as operator+ is the operand to the right hand side of the operator.

   ```
   type operator sign(type & rhs)
   {/* ... body ... */}
   ```

2. as a non-member function:
   Operator function takes the object of the class as first argument.

   ```
   type operator sign(type &lhs, type &rhs)
   {/* ... body*/}
   ```

Assignment operator overload


```
Frac& Frac::operator=(const Frac &source)
{
    m_Numerator = source.m_Numerator;
    m_Denominator = source.m_Denominator;
    return (*this);
}


// h assigned to g, which is assigned to f

Fraction f, g, h;
f = g = h;
```

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates
2 Tricks and tools

Function Templates


```
#include <iostream>
using namespace std;
template <class T>
T sum(T a, T b){
  T result;
  result = a + b;
  return result;
}
int main(){
  int i=5, j=6, k;
  double f=2.0, g=0.5, h;
  k=sum<int>(i,j); // <int> needed?
  h=sum<double>(f,g);
  cout << k << endl;
  cout << h << endl;
  return 0;
}
```

Function templates


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

template <typename T, typename U>
bool are_equal(T a, U b)
{
  return (a==b);
}

int main ()
{
  if(are_equal(10,10.0))
    cout << "x and y are equal\n";
  else
    cout << "x and y are not equal\n";
  return 0;
}
```

The general form of a template definition

General syntax for a Function (review)

```
template <typename type>
ret-type func-name(parameter list)
{
   // body of function
}
```

General syntax for a class

```
template <typename type>
class class-name
{
  //class contents
    .
    .
}
```

Template classes with multiple files

Recall:

- '.h' file – where you write the template class definitions and include the below hpp at the end
- '.hpp' file – the template class function definitions
- '.cpp' file(s) – one or more, main, etc.

Outline

1 C++
Hello world
Variables
Operators
Input and output
Conditionals
Loops
Functions
Multiple files
Scope
Arrays
Structs
Classes
Constructors
Overloading
Templates

2 Tricks and tools

Development

- Discuss: Codeblocks, lab computers, etc
- Tip 1: How to debug with cout statements
- Tip 2: How to write and test only little parts of code at a time

We'll come back to these throughout the semester and in lab