Scientific Programming With C++

Jerry Ebalunode

http://support.hpedsi.uh.edu/

University of Houston
Houston, TX

UNIVERSITY of HOUSTON
DIVISION OF RESEARCH
HEWLETT PACKARD ENTERPRISE DATA SCIENCE INSTITUTE

- Writing a Basic C++ Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Operators
- Control Structures
- Functions in C++
- Arrays
- Pointers
- Working with Files

Control Structures

Selection Structure used for branching

Loop Structure used for iteration or repetition

Sequence Structure is a sequence of statements

Selection Structure
Conditional Expressions

- Using **if-else**

```
if  (a  > b)
{
    z = a;
}
else
{
    z = b;
}
// equivalent to:
z = max (a, b)
```

- or ternary operator (**?:**)

```
z = (a  > b)  ? a : b ;
```

If-else: Logical Expressions

```
if (temp > 75  && temp < 80)
{
  cout  <<  "It's nice  weather outside\n";
}

if  (value  ==  'e' ||  value ==  'n' )
{
cout  <<  "Exiting  the program.\n";
}
else
{
cout  <<  "\nIn the program.\n";
}
```

Loop Structures

- For repeating a sequence of steps/statements
- The statements in a loop are executed a specific number of times, or until a certain condition is met
  - Three types of loops
    - **`for`**
    - **`while`**
    - **`do-while`**

`for` Loop

```
for (start_value; end_condition; stride)
        statement;

for (start_value; end_condition; stride)
{
        statement1;
        statement2;


        statementN;
}
```

`for` Loop, C++11 auto

Read only access, call by value

```
for(auto item :  array_iterable_sequence)
{
      cout<< item <<endl;
}
```

`for` Loop, C++11 auto

Read and write access, call by reference

```
for (auto & item :  array_iterable_sequence)
        item += new_value;


for (auto & item :  array_iterable_sequence)
{
        cout<< item <<endl;
}
```

`for` Loop Example

```
#include  <iostream>
using namespace std;
int main()
{
  int i;
  for ( i=0;  i<3;  i++)
     cout <<  “What a wonderful class!\n”;
return  0;
}
```

Output:

What a wonderful class!
What a wonderful class!
What a wonderful class!

for Loop C++11 auto Example

```
#include  <iostream>
using namespace std;
int main()
{
  int  myarray[]={1,2,3};
  for ( auto item : myarray)
     cout << item<<  “  What a wonderful class!\n”;
return  0;
}
```

Output:
1 What a wonderful class!
2 What a wonderful class!
3 What a wonderful class!

`for` Loop C++11 auto
Call by value Example

```
int  myarray[]={1,2,3};
for ( auto item : myarray) item *=2;

for ( auto item : myarray)
cout << item<<  “  What a wonderful class!\n”;
```

Output:
1 What a wonderful class!
2 What a wonderful class!
3 What a wonderful class!

`for` Loop C++11 auto
Call by reference Example

```
int  myarray[] = {1,2,3};
for ( auto & item : myarray) item *=2;

for ( auto item : myarray)
cout << item<<  “  What a wonderful class!\n”;
```

Output:
2 What a wonderful class!
4 What a wonderful class!
6 What a wonderful class!

`for` Loop and break keyword

Example: forLoop.cpp

```
#include  <iostream>
using namespace std;
int main()
{
  int i;
  for ( i=0;  i<=10;  i=i+2)
  {
    if  (i >5) {  break;  }
    cout <<  "What a wonderful class!\n";
}
  return  0;
}
```

Output:
What a wonderful class!
What a wonderful class!
What a wonderful class!

`break` is the keyword used to **stop** the loop in which it is present

`while` Loop

- The while loop can be used if you don't know how many times a loop should run

```
while (condition_is_true)
{
   statement (s);
}
```

- The statements in the loop are executed till the loop condition is no longer true

- The condition that controls the loop can be modified inside the loop (this is true in the case of **for loops** too!)

`while` Loop Example: whileLoop.cpp

```
#include  <iostream>
using namespace std;
int main()
{
  int counter,  value;
  value = 5;
  counter = 0;
  while ( counter < value)
  {
    counter++;
    cout <<  "counter  value is: " <<  counter <<  endl;
  }
  return  0;
}
```

Output:

```
counter value is: 1
counter value is: 2
counter value is: 3
counter value is: 4
counter value is: 5
```

Functions in C++ Language

- Functions are self-contained blocks of statements that perform a specific task
- Written once and can be used multiple times
  - Promote code reuse
  - Makes code maintenance easy
- Two types of functions
  - Standard Library
  - User-Defined
- Like operators, C++ functions can be overloaded too

Categories of Functions

- Functions that take input and return output
- Functions that take no input, and return no output

- Functions that take input and use it but return no output
- Functions that take no input but return output

Standard Functions

- These functions are provided to the user in library files
- In order to use the functions, the user should include the appropriate library files containing the function definition
- For example, following functions are available through the math library named <cmath>
  - ceil(x)
  - cos(x)
  - exp(x)
  - log(x)
  - floor(x)
- All these functions take **double** values

Standard Function

Example: mathExample1.cpp

```
#include  <iostream>
#include  <cmath>
using namespace std;
int main()
{
        double  x = 0;
        cout  <<  "Enter  a double  value\n";
        cin >>  x;
        cout  <<"Square root  of  " <<  x <<  " is  " <<  sqrt(x);
        cout  <<  "\nLog  of  " <<  x <<  " is  " <<  log(x)<<  endl;
        return  0;
}
```

Note that the math library header is included

Standard functions available through math library

Output
Enter a double value
2.0
Square root of 2 is 1.41421
Log of 2 is 0.693147

User-Defined Function-using prototypes
Example: noInputNoReturn.cpp

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

void add();

int main()
{
     add() ;
     add() ;
    return  0;
}
void add()
{
        int a,  b,  c;
       cout  <<  "\n Enter Any 2 Numbers : ";
       cin >>  a >>  b;
       c = a + b;
       cout  <<  "\n Addition  is  : " <<  c;
}
```

Function Prototype or Declaration useful when the function is invoked before its definition is provided

Invoking the function add

Function Definition

Output:

```
Enter Any 2 Numbers : 1 2
Addition  is  : 3
Enter Any 2 Numbers : 4 5
Addition  is  : 9
```

Guidelines For Sending Input Values To Functions

- Determine the number of values to be sent to the function
- Determine the data type of the values that needs to be sent
- Declare variables having the determined data types as an argument to the function
- Use the values in the function
- Prototype the function if its definition is not going to be available before the place from where it is invoked
- Send the correct values when the function is invoked

Passing Values to Functions
Example: passValue1.cpp

```
#include  <iostream>
using namespace std;
void add( int  x, int y);
int main()
{
        int a,  b;
       cout  <<  "\n Enter Any 2 Numbers : ";
       cin >>  a >>  b;

       add ( a, b );
       return  0;
}
void add(int a, int b)
{
                int  c;
                 c= a + b;
                cout  <<  "\n Addition  is  : " <<  c <<endl;
}
```

function prototype: int ? , int ?

Actual parameters: a, b

Formal parameters: a, b

Note: The variables used as formal and actual parameters can have different names.

Passing Values to Functions from int main:

Example: passValue2.cpp

Note that the cstdlib library header is included

add(int a, int b) function returns integer (int)

Notice that main has two arguments

argc == argument count

argv == 2D array to store the arguments data

```
#include  <iostream>
#include <cstdlib>
using namespace std;
int add( int a, int b);

int main( int argc, char ** argv)
{
        int a,  b, c;
        if  (argc != 3) {
        cout  <<  "\nInsufficient num.  of  arguments.\n";
        cout  <<  "\nUsage:"  <<  argv[0] <<  " <firstNum>  <secondNum>\n";
        } else{
        a = atoi(argv[1]);
        b = atoi(argv[2]);
        c  = add(a, b);
        cout  <<  "\n Addition  of  a and b is  : " <<  c <<endl;
        }
        return  0;
}
int add(int a, int b)
{
   return (a + b);
}
```

argv[1] holds the first number

argv[2] holds the second number

The atoi function converts the keyboard input/arguments, which is a string, into integer. It is part of the cstdlib library

Functions Calling Functions

- We’re already doing this!
  - main() IS a function!
- Only requirement:
  - Function’s declaration must appear first
- Function’s definition typically elsewhere
  - After main()"s definition
  - Or in separate file
- Common for functions to call many other functions
- Function can even call itself → "Recursion"

Boolean Return-Type Functions

- Return-type can be any valid type
  - Given function declaration/prototype:
    bool appropriate(int rate);
  - And function's definition:

```
bool appropriate (int rate)
{
        return (((rate>=10)&&(rate<20))||(rate==0);
}
```

  - Returns "true" or "false"
  - Function call, from some other function:

```
if (appropriate(entered_rate))
    cout << "Rate is valid\n";
```

Function Overloading (or Polymorphism)

- Overloading refers to the use of same thing for different purposes
- Function overloading means that we can use the same function name to create functions that perform a variety of different tasks
- The function names are same but the signature is different – that is, different return type, different argument lists
- Example

```
int add(int a, int b);
int add(int a, int b, int c);
double add(double a, double b);
```

Function Overloading Example: fctOverloading.cpp (1)

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

//overloading volume
int volume (int);  //prototype declaration
double volume (double, double); //prototype declaration
double volume (double, double, double); //prototype decl.

int main(){
 cout << "cube vol: "<< volume(10) << endl;
 cout << "cylinder vol: " << volume(2.5, 8.5) << endl;
 cout << "cuboid vol: " << volume(100.5, 75.5, 15.5)<<"\n";
 return 0;
}
```

…

Function Overloading Example: fctOverloading.cpp(2)

```
…
//volume of a cube
int volume(int s){
  return s*s*s;
}

//volume of a cylinder
double volume(double r, double h){
  return (3.14519 * r * r * h);
}

//rectangular box or cuboid
double volume(double l, double b, double h){
  return (l*b*h);
}
```

Output

```
cube vol: 1000
cylinder vol: 167.088
cuboid vol: 117610
```

Function Templates

- If the program logic and operations are identical for each data type, overloaded functions can be written more compactly using function templates
- A single function template definition is written
- By a single function template, you can define the whole family of overloaded functions

Function Templates: fctTemplate.cpp (1)

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

template <class T>
T maximum(T value1, T value2, T  value3){
  T maxValue = value1;
  if (value2 > maxValue){
    maxValue = value2;
  }
  if(value3 > maxValue){
    maxValue = value3;
  }
  return maxValue;
}
```

…

Function Templates: fctTemplate.cpp (2)

```
…

int main(){
  int val1, val2, val3;
  double val4, val5, val6;
  cout << "\nEnter three integer values\n";
  cin >> val1 >> val2 >> val3;
  cout << "Maximum integer value is: "<< maximum(val1,val2,val3);

  cout << "\nEnter three double values\n";
  cin >> val4 >> val5 >> val6;
  cout <<"Maximum double value is: "<< maximum(val4,val5,val6);
  return 0;
}
```

Function Templates: fctTemplate.cpp (3)

Output:

```
Enter three integer values
2 3 4
Maximum integer value is: 4
Enter three double values
2.1 3.1 1.1
Maximum double value is: 3.1
```

Introduction to File Input

- We can use cin to read from a file in a manner very similar to reading from the keyboard
- Only an introduction is given here, more details are in chapter 12
  - Just enough so you can read from text files and process larger amounts of data that would be too much work to type in

Opening a Text File

- Add at the top

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

- You can then declare an input stream just as you would declare any other variable.

```
ifstream inputStream;
```

- Next you must connect the inputStream variable to a text file on the disk.

```
inputStream.open("filename.txt");
```

- The “filename.txt” is the pathname to a text file or a file in the current directory

Reading from a Text File

- Use

```
inputStream >> var;
```

- The result is the same as using `cin >> var` except the input is coming from the text file and not the keyboard
- When done with the file close it with

```
inputStream.close();
```

File Input Example (1 of 2)

- Consider a text file named player.txt with the following text

Display 2.10 Sample Text File, **player.txt**, to Store a Player's High Score and Name

```
100510
Gordon Freeman
```

File Input Example (2 of 2)

Display 2.11 Program to Read the Text File in Display 2.10

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>

4  using namespace std;
5  int main( )
6  {
7      string firstName, lastName;
8      int score;
9      fstream inputStream;

10     inputStream.open("player.txt");

11     inputStream >> score;
12     inputStream >> firstName >> lastName;

13     cout << "Name: " << firstName << " "
14          << lastName << endl;
15     cout << "Score: " << score << endl;
16     inputStream.close();

17     return 0;
18 }
```

Sample Dialogue

```
Name: Gordon Freeman
Score: 100510
```