C++

C++

C++ is an object-oriented programming language that is widely supported on supercomputing platforms. Major portions of the LFM are written in C++ to take advantage of a library (P++) used to distribute LFM data for parallel computing.

This tutorial is designed to quickly bring Fortran developers up to speed on C++. It should prepare you to start working with A++ & P++ and introduce you to the coding style of the LFM.

Part 1: The Basics

Control Structures

control.C: An exceedingly fast tour through C++ Control Structures.

control.C

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

void forLoop()
{
int array[5] = {1,2,3,4,5};
cout << +FUNCTION+ << endl;
for (int i=0; i < 10; i++){
    cout << array[i] << "\t";
  }
cout << endl;
}

void whileLoop()
{
int accountBalance = 100;
cout << +FUNCTION+ << endl;

int iterations = 0;
while (accountBalance > 0){
    // Buy some stuff
    accountBalance = accountBalance - 12;
    iterations++;
  }
cout << "It took " << iterations << " iterations to spend everything\!" << endl;
}

void conditionals()
```

```
{
cout << +FUNCTION+ << " (ie. \"if\", \"else\")" << endl;

float a;
cout << "Enter a number" << endl;
cin >> a;

if (a == 42){
    cout << "Type this into Google: \"answer to life, the universe and everything\"." << endl;
  }
else if (a != 42){
    cout << "a is not 42." << endl;
  }
else {
    cout << "catch-all else statement." << endl;
  }
}


int main ()
{
cout << "Select a control structure:" << endl;
cout << "1.  for" << endl
<< "2.  while" << endl
<< "3.  if" << endl;

int controlSelection = 0;
cin >> controlSelection;

switch (controlSelection) {
  case 1:
    forLoop();
    break;
  case 2:
    whileLoop();
    break;
  case 3:
    conditionals();
  default:
    cout << "Did not understand control selection.  Try again." << endl;
  }

return 0;
}
```

Functions: Pass By Value vs Reference

functions.C; Learn about overloading, recurssion, inline, etc. at this C++ tutorial.

functions.C

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

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

int main ()
{
  int x=1, y=3, z=7;
  cout << "Before duplication:" << endl;
  cout << "x=" << x << ", y=" << y << ", z=" << z;
  duplicate (x, y, z);
  cout << "After duplication:" << endl;
  cout << "x=" << x << ", y=" << y << ", z=" << z;
  return 0;
}
```

Arrays

arrays.C: Arrays as function parameters.

arrays.C

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

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

int main ()
{
int firstarray[] = {5, 10, 15};
int secondarray[] = {2, 4, 6, 8, 10};
printarray (firstarray,3);
printarray (secondarray,5);
return 0;
}
```

Exercise: Trapezoid Integration

Implement a function to compute trapezoidal integration for

```
f(x) = sin(x)
```

The function prototype should look like:

```
float trapInt(int a, int b, int n);
```

Demonstrate that it works properly:

```
trapInt(0, 3.141592, 100)
```

should return a number close to 2.0.

Part 2: Advanced Topics

Pointers

This section was inspired by the cplusplus.com tutorial on pointers and Practical C++ Programming

Introduction

pointers.C

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

int main(int argc, char **argv)
{
  float number = 42;

  // "&" is the reference operator
  float *number_ptr = &number;

  cout << "Number      = " << number << endl
       << "Number_ptr  = " << number_ptr << endl
       << "*Number_ptr = " << *number_ptr << endl
       << "----" << endl;

  // "*" is the dereference operator.
  *number_ptr = 10;

  cout << "Number      = " << number << endl
       << "Number_ptr  = " << number_ptr << endl
       << "*Number_ptr = " << *number_ptr << endl;

  return 0;
}
```

Definitions/review

- A **pointer** is an address for a "thing". Relate this to every day life: You live in a house (a "thing"), and its street address might be "123 W. Main Street". An address is a small thing that can be written on a piece of paper. Putting a house on a piece of paper is something else, requiring a lot of work and a very large crane!
- *** is the dereference operator: Given a pointer, get the thing referenced
- **&** is the reference operator: Get the address of a given thing.
- **NULL** is a special pointer that points to nothing (actual numeric value is 0). We will come back to this in regards to dynamically allocated memory.

Arrays and Pointers

arrayPointers.C

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

int main(int argc, char **argv)
{
float even[4] = {2.0, 4.0, 6.0, 8.0};

// Note I do not have a dereferencing operator!
float *even_ptr = even;

for (int i=0; i < 4; i++){
    cout << "even[__ i __]: " << even[i] << "\t"
         << "even_ptr[__ i __]: " << even_ptr[i] << "\t"
         << "*even_ptr + " << i << ": " << *(even_ptr+i) << endl;
  }

return 0;
}
```

Dynamic allocation

What happens when we need to change the size of an array in a program?

dynamicArrays.C

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

int main(int argc, char **argv)
{
  cout << "Dynamically allocating an array of 2 elements" << endl;
  int *array_ptr = new int[2];
  array_ptr[0] = 1;
  array_ptr[1] = 2;

  cout << "Array contents:" << endl;
  for (int i=0; i < 4; i++){
    cout << array_ptr[i] << "\t";
  }
  cout << endl;

  cout << "On second thought, we need an array of 4 elements.  Deallocating memory..." << endl;

  cout << "Array contents:" << endl;
  for (int i=0; i < 4; i++){
    cout << array_ptr[i] << "\t";
  }
  cout << endl;

  cout << "... allocating a bigger array!" << endl;
  delete [] array_ptr;
  array_ptr = NULL;
  array_ptr = new int[4];
  array_ptr[0] = 1;
  array_ptr[1] = 2;
  array_ptr[2] = 3;
  array_ptr[3] = 4;

  cout << "Array contents:" << endl;
  for (int i=0; i < 4; i++){
    cout << array_ptr[i] << "\t";
  }
  cout << endl;

  delete [] array_ptr;

  return 0;
}
```

Warning: It's extremely easy to shoot yourself in the foot with dynamically-allocated memory!

Find more on the CPlusPlus.com tutorial on Dynamic Memory

Row vs column ordering (compare to Fortran)?

Exercise: Trapezoid integration (continued)

Write a program that asks the user to input the maximum number of intervals to use when computing Trapezoid integration for `sin` between 0 and pi. Store the results of trapezoid integration in an array for all values of n, n-1, n-2, ..., 1 intervals. Print the contents of this array to the user.

Classes

Introduction

Definitions:

- A **class** is an expanded concept of a data structure: it can hold both data and functions.
- An **object** is an instantiation (or instance) of a class.

Data and functions can be

- Public: Anyone can access
- Private: Only class methods can access (encapsulation!)
- Protected: a special case of private

Learn more definitions at the cplusplus.com Classes I tutorial

```
#include <iostream>
#include <math.h>

class point
{
public:
// Constructor
point(const float &a, const float &b, const float &c) { x=a; y=b; z=c;};

float x,y,z;
};

float distance(const point &p1, const point &p2)
{
  return sqrt( fabs(pow(p2.x - p1.x, 2) +
                    pow(p2.y - p1.y, 2) +
                    pow(p2.z - p1.z, 2) ));
}

int main(int argc, char **argv)
{
  point origin(0, 0, 0);
  point myPoint(2, 2, 2);

  std::cout << "The distance is " << distance(myPoint, origin) << std::endl;
  return 0;
}
```

Operator Overloading

Learn more at the Cplusplus.com "Classes 2" Tutorial.

```
point point::operator *(const float &scalar)
{
  return point(x*scalar, y*scalar, z*scalar);
}

...

   std::cout << "myPoint is " << myPoint.x << " " << myPoint.y << " " << myPoint.z << std::endl;
   point p5 = myPoint*5;
   std::cout << "myPoint*5 is " << p5.x << " " << p5.y << " " << P5.z << std::endl;
```

Inheritance, derived classes, etc.

Advanced topic that does not appear to be extensively used by A++ & P++. However, the LFM has a few examples of derived classes:

- I/O routines: Regardless of file format, the code needs to output a small set of vectors.
- Interfaces between codes using InterComm or file exchanges

Exercise: Vector class

Implement a vector class that can compute the dot product and cross product of two vectors.

Answers to Exercises

No cheating! 🙂

- Trapezoid Integration

trapezoid.C

```
#include <math.h>
#include <iostream>

using namespace std;

float trapInt(float a, float b, int n)
{
float dx = (b-a) / float(n) ;

float result = 0.0;
for (int i=0; i < n-1; i++){
    float x0 = a+dx*float(i);
    float x1 = a+dx*float(i+1);

    result += dx*(sin(x1)+sin(x0));
  }

return (result / 2.0);
}

int main(int argc, char **arv)
{
  cout << "The Area is " << trapInt(0.0, 3.141592, 100) << endl;
  return 0;
}
```

- Pointers: Dynamic allocation & trapezoid integration

- Classes: Vector Class

  vector1.C

```
#include <iostream>
#include <math.h>

using namespace std;

class Vector
{
public:
  float v0, v1, v2;

  Vector(const float &v0, const float &v1, const float &v2);

  float length() const;
  Vector direction() const;
  float dot(const Vector &inVector);
};

Vector::Vector(const float &v0, const float &v1, const float &v2)
{
  this->v0 = v0;
  this->v1 = v1;
  this->v2 = v2;
}

float Vector::length() const
{
  return sqrt( v0*v0 + v1*v1 + v2*v2 );

}

Vector Vector::direction() const
{
  float vecLength = this->length();

  float ihat = v0 / vecLength;
  float jhat = v1 / vecLength;
  float khat = v2 / vecLength;

  return Vector(ihat, jhat, khat);
}

float Vector::dot(const Vector &inVector)
{
  return ( inVector.v0 * v0 +
           inVector.v1 * v1 +
           inVector.v2 * v2 );
}


int main()
{

  Vector test(2.0, 0.0, 0.0);

  cout << "The vector is: " << test.v0 << " " << test.v1 << " " << test.v2 << endl;
  cout << "Its length is: " << test.length() << endl;

  Vector direction(1,1,1);
  direction = test.direction();

  cout << "Vector direction is " << direction.v0 << " " << direction.v1 << " " << direction.v2 << endl;

  cout << "dot product with unit vector is " << test.dot(direction) << endl;

  return 0;
}
```

Alternative implementation:

vector2.C

```
#include <iostream>
#include <math.h>

using namespace std;

class myVector
{
public:
// Constructor
myVector(const float &a, const float &b, const float &c) { x=a; y=b; z=c;};

float dot(const myVector &v);
myVector cross(const myVector &v);

float x,y,z;
};

float myVector::dot(const myVector &v)
{
  return (x*v.x + y*v.y + z*v.z);
}

myVector myVector::cross(const myVector &v)
{
  return myVector(y*v.z - v.y*z,
                  x*v.z - v.x*z,
                  x*v.y - v.y*x);
}

ostream &operator << (ostream &outs, const myVector &v)
{
  outs << v.x << ", " << v.y << ", " << v.z << endl;
}

int main(int argc, char **argv)
{
  myVector jhat(0, 1, 0);
  myVector khat(0, 0, 1);

  std::cout << "jxk = " << jhat.cross(khat) << std::endl;
  std::cout << "j.k = " << jhat.dot(khat) << std::endl;

  return 0;
}
```

Resources

- C++ Tutorial - Learn the basics!
- C++ Reference - Documentation (and examples) for all C++ standard libraries.
- Q&A - One of the best computer science question-and-answer websites out there.
- C++ FAQ Lite - Answers to the most frequently asked C++ questions. Also serves as a good reference to some of the trickier aspects of C++, such as const correctness.