Introduction to C++

P. Bansal, A. Dabrowski

Values, pointers and references

Classes and Inheritance

Templates

What you must know beforehand?

- Conditional structures: if, else
- Iterative loops: for, while, do while
- Functions and Objects
- A basic understanding of the C++ type system

Values, pointers and references

What is the output?

```
#include<iostream>

void f1(int x) { x = 10; }
void f2(int* x) { *x = 15; }
void f3(int& x) { x = 20; }

int main() {
  int x = 5;
  f1(x); std::cout<< x <<std::endl;
  f2(&x); std::cout<< x <<std::endl;
  f3(x); std::cout<< x <<std::endl;
}
```

Classes and Inheritance

```
class Polygon {
public:
  Polygon() {} //default Constructor
  ~Polygon() {} //default Destructor

  Polygon(double w, double h) :
    width(w), height(h) {} //Constructor 2

  void set(double w, double h) {//member func
    width = w;
    height = h;
  }
private: //protected
  double width, height;
};
```

- public: can be accessed by anybody
- private: can only be accessed by class members
- protected: can only be accessed by class members, derived and friend classes

How to create an object of class Polygon?

```
// static
Polygon S;
S.set(5.0,4.0)
//or
Polygon S(5.0,4.0);

// dynamic
Polygon *S; // pointer to Polygon object

S = new Polygon;
S->set(5.0,4.0);
//or
S = new Polygon(5.0,4.0);
delete S;
```

Derived Classes

```
class Triangle : public Polygon {
public:
  double area () {
    return (width*height/2);
  }
};

class Rectangle : public Polygon {
public:
  double area () {
    return (width*height);
  }
};
```

```
Triangle S1;
Rectangle S2;

S1.set(5.0,4.0);
S2.set(5.0,4.0);

std::cout << S1.area() << std::endl;
std::cout << S2.area() << std::endl;
```

What is the output?

If

```
class Rectangle : private Polygon {..}
// or
class Rectangle : protected Polygon {..}
```

will it work?

Virtual Functions

```
class Base1 {
public:
  void nonVirtualFn() {
    cout<<" Non-virtual func in Base1 "<<endl;
  }
  virtual void virtualFn() {
    cout<<" Virtual func in Base1 "<<endl;
  }
};

Base1 *bBase = new Base1;
bBase->nonVirtualFn();
bBase->virtualFn();
```

What is the output?

```
class Derived : public Base1{
public:
 void nonVirtualFn() {
    cout<<" Non-virtual func in Derived "<<endl;
 }
 void virtualFn() {
    cout<<" Virtual func in Derived "<<endl;
 }
};

Base1 *bDerived = new Derived;
bDerived->nonVirtualFn();
bDerived->virtualFn();
```

What is the output?

```
class Base2 { //Abstract Base Class
public:
  virtual void pureVirtualFn() = 0;
};

class Derived : public Base1, public Base2 {
public:
  ....

  void pureVirtualFn () {
    cout<<"Pure virtual func defination"<<endl;
  }
};
```

- An object of an abstract class cannot be created!
- Used for interfacing

Templates

```
int product(int a, int b) {
  int c = a*b;
  cout<<"a*b = "<< c <<endl;
  return c;
}

double product(double a, double b) {
  int c = a*b;
  cout<<"a*b = "<< c <<endl;
  return c;
}
```

Function templates

```
template<typename T> //or template<class T>
T productTemp1(T a, T b) {
  T c = a*b;
  cout<<"a*b = "<< c <<endl;
  return c;
}
int a1 = 9; int b1 = 7;
double a2 = 2.5; double b2 = 3.5;

productTemp1(a1,b1);
productTemp1(a2,b2);
productTemp1(a1,b2);//does it work?

// explicit instantiation, good practice
productTemp1<int>(a1,b1);
productTemp1<double>(a2,b2);
productTemp1<double>(a1,b2);//does it work?
```

A more flexible template using multiple types

```
template<typename T1, typename T2, typename T3>
T3 productTemp2(T1 a, T2 b) {
   T3 c = a*b;
   cout<<"a*b = "<< c <<endl;
   return c;
}
productTemp2<int,double,double>(a1,b2);
productTemp2<double,int,double>(a2,b1);
```

Class templates

```
template<typename T> //or template<class T>
class System {
public:
  System (T inData) : data(inData) {}

  void setData (T inData) {
    data = inData;
  }
  T getData () {
    return data;
  }
  void processData () {
    if (data >= 1)
      cout<<"This doesn't look good."<<endl;
  }
private:
  T data;
}
```

```
// avg. temperature rise, global warming
double x = 1.0;

// only explicit instantiation is possible
System<double> earth(x);
earth.processData();
```

- Multiple template parameters can be used.
- Also possible to pass an object.

For detailed information:

- http://en.cppreference.com
- http://www.cplusplus.com/
- http://www.geeksforgeeks.org/inheritance-in-c/
- https://www.codeproject.com/Articles/257589/An-Idiots-Guide-to-Cplusplus-Templates-Part