June 13th, 2013

1. In C++, interfaces are implemented using abstract classes. A class is made abstract by declaring at least one of its functions as pure virtual function. A pure virtual function is specified by placing “=0” in its declaration.
2. Derived classes can access the protected data in its parent class.
3. A derived class inherits all base class methods with the following exceptions:
4. Constructors, destructors and copy constructors of the base class.
5. Overloaded operators of the base class.
6. The friend functions of the base class.
7. When using a public inheritance, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class’s private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.
8. Derive from multiple classes:

Class Rectangle: public Shape, public PaintCost

1. <http://www.tutorialspoint.com/index.htm> is a really useful webpage.
2. Pointers:
3. Definition: type \*var-name;

e.g. int \*ip;

int var = 20;

int \*ip;

ip = &var;

Pointer to pointer: int \*\*var;

1. Passing pointers to functions:

Declare the function parameter as a pointer type:

Returntype functionName (type \*var)

And now var need to be a pointer which like &value.

#include <stdio.h>

#include <time.h>

void getSeconds(unsigned long \*par);

int main ()

{

unsigned long sec;

getSeconds( &sec );

/\* print the actual value \*/

printf("Number of seconds: %ld\n", sec );

return 0;

}

void getSeconds(unsigned long \*par)

{

/\* get the current number of seconds \*/

\*par = time( NULL );

return;

}

1. The function which can accept a pointer can also accept an array as shown:

The name of the array can be considered as the first address of the array.

#include <stdio.h>

/\* function declaration \*/

double getAverage(int \*arr, int size);

int main ()

{

/\* an int array with 5 elements \*/

int balance[5] = {1000, 2, 3, 17, 50};

double avg;

/\* pass pointer to the array as an argument \*/

avg = getAverage( balance, 5 ) ;

/\* output the returned value \*/

printf("Average value is: %f\n", avg );

return 0;

}

double getAverage(int \*arr, int size)

{

int i, sum = 0;

double avg;

for (i = 0; i < size; ++i)

{

sum += arr[i];

}

avg = (double)sum / size;

return avg;

}

1. Returning a pointer from a function:
2. int \* myFunction()
3. {
4. .
5. .
6. .
7. }

It is not good idea to return the address of a local variable to outside of the function so you would have to define the local variable as static variable.

1. Throw exceptions:

#include <iostream>

using namespace std;

double division(int a, int b)

{

if( b == 0 )

{

throw "Division by zero condition!";

}

return (a/b);

}

int main ()

{

int x = 50;

int y = 0;

double z = 0;

try {

z = division(x, y);

cout << z << endl;

}catch (const char\* msg) {

cerr << msg << endl;

}

return 0;

}

1. Namespace: is designed to overcome the difficulty that with the same name available in different libraries. It uses additional information to differentiate similar functions, classes, variables etc. Using namespace, you can define the context in which names are defined. I essence, a namespace defines a scope.

Definition:

namespace namespace\_name {

// code declarations

}

Call the namespace-enabled version of function or variable:

name::code; // code could be variable or function.

e.g.

#include <iostream>

using namespace std;

// first name space

namespace first\_space{

void func(){

cout << "Inside first\_space" << endl;

}

}

// second name space

namespace second\_space{

void func(){

cout << "Inside second\_space" << endl;

}

}

int main ()

{

// Calls function from first name space.

first\_space::func();

// Calls function from second name space.

second\_space::func();

return 0;

}

Result:

Inside first\_space

Inside second\_space

The using directive can also be used to refer to a particular item within a namespace. For example, if the only part of the std namespace that you intended to use is cout, you can refer to it as follows:

#include <iostream>

using std::cout;

int main ()

{

cout << "std::endl is used with std!" << std::endl;

return 0;

}

Nested namespaces:

You can define one namespace inside another namespace:

namespace namespace\_name1 {

// code declarations

namespace namespace\_name2 {

// code declarations

}

}

And you can access members of nested namespace by using result ion operators as follow:

// to access members of namespace\_name2

using namespace namespace\_name1::namespace\_name2;

// to access members of namespace:name1

using namespace namespace\_name1;

1. Python: CGI, common gateway interface, is a set of standards that define how information is exchanged between the webserver and a custom script. It is a standard for external gateway programs to interface with information servers such as HTTP servers.