1. **Friend Functions**

www.hndit.com

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

**class**MyClass {
  **int**a, b;
**public**:
  MyClass(**int**i, **int**j) { a=i; b=j; }
  friend **int**friendFunction(MyClass x); // a friend function
};

// friendFunction() is a not a member function of any class.
**int**friendFunction(MyClass x)
{
  /\* Because friendFunction() is a friend of MyClass, it can
     directly access a and b. \*/
  **int**max = x.a < x.b ? x.a : x.b;

  **return**max;
}
**int**main()
{
  MyClass n(18, 111);
  cout << "friendFunction(n) is " << friendFunction(n) << "\n";
  **return**0;
}

1. Friend functions can be shared by two or more classes

#include <iostream>
**using**namespace std;
**class**Cylinder; // a forward declaration
**enum**colors { red, green, yellow };
**class**Cube {
 colors color;
**public**:
  Cube(colors c) { color = c; }
  friend bool sameColor(Cube x, Cylinder y);
};

**class**Cylinder {
 colors color;
**public**:
  Cylinder(colors c) { color= c; }
  friend bool sameColor(Cube x, Cylinder y);
};

bool sameColor(Cube x, Cylinder y)
{
  **if**(x.color == y.color)
     **return true**;
  **else**
     **return false**;
}

**int**main()
{
  Cube cube1(red);
  Cube cube2(green);
  Cylinder cyl(green);

  **if**(sameColor(cube1, cyl))
    cout << "cube1 and cyl are the same color.\n";
  **else**
    cout << "cube1 and cyl are different colors.\n";

  **if**(sameColor(cube2, cyl))
    cout << "cube2 and cyl are the same color.\n";
  **else**
    cout << "cube2 and cyl are different colors.\n";

  **return**0;
}

1. A function can be a member of one class and a friend of another

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

**class**Cylinder; // a forward declaration

**enum**colors { red, green, yellow };

**class**Cube {
 colors color;
**public**:
  Cube(colors c) { color= c; }
  bool sameColor(Cylinder y);
};
**class**Cylinder {
 colors color;
**public**:
  Cylinder(colors c) { color = c; }
  friend bool Cube::sameColor(Cylinder y);
};
bool Cube::sameColor(Cylinder y) {
  **if**(color == y.color)
    **return true**;
  **else**
    **return false**;
}
**int**main()
{
  Cube cube1(red);
  Cube cube2(green);
  Cylinder cyl(green);

  **if**(cube1.sameColor(cyl))
    cout << "cube1 and cyl are the same color.\n";
  **else**
    cout << "cube1 and cyl are different colors.\n";

  **if**(cube2.sameColor(cyl))
    cout << "cube2 and cyl are the same color.\n";
  **else**
    cout << "cube2 and cyl are different colors.\n";

  **return**0;
}

1. Constrictors and destructors

#include <iostream>
**using**namespace std;
**class**MyClass {
**public**:
  **int**x;

  MyClass(**int**i);  // constructor
  ~MyClass();      // destructor
};
// Implement a parameterized constructor.
MyClass::MyClass(**int**i) {
    x = i;
}
// Implement MyClass destructor.
MyClass::~MyClass() {
  cout << "Destructing object whose x value is " << x  <<" \n";
}
**int**main() {
  MyClass t1(5);
  MyClass t2(19);

  cout << t1.x << " " << t2.x << "\n";
   **return**0;
}

1. Life Cycle of objects

www.hndit.com

#include<iostream.h>

class Test{

public:

Test(){ cout<<"Constructor Invoked "<<endl;}

~Test(){cout<<"Destructor Invoked "<<endl;}

};

Test obj1;

int main()

{cout<<"main() BEGINS"<<endl;

Test obj2;

{cout<<"INNER BLOCK BEGINS"<<endl;

Test obj3;

cout<<"INNER BLOCK ENDS"<<endl;}

cout<<"main() ENDs"<<endl;

return 0;

}

1. namespace & using

*#include <iostream>*

*using* *namespace* std;

*namespace* first

{

 *int* x = 5;

 *int* y = 10;

}

*namespace* second

{

 *double* x = 3.1416;

 *double* y = 2.7183;

}

*int* main () {

 *using* first::x;

 *using* second::y;

 cout << x << endl;

 cout << y << endl;

 cout << first::y << endl;

 cout << second::x << endl;

 *return* 0;

}