C++

Reference variable

- A reference variable is a "reference" to an existing variable, and it is created with the & operator:#include <iostream>

```
#include <string>
using namespace std;
int main() {
  string food = "Pizza";
  string &meal = food;
  cout << food << "\n";
  cout << meal << "\n";
  return 0;
}
```

Memory address

- n the example from the previous page, the & operator was used to create a reference variable.
- But it can also be used to get the memory address of a variable

- string food = "Pizza";
- cout << &food; // Outputs 0x6dfed4

Parameter passing by reference

```
void swapNums(int &x, int &y) {
  int z = x;
  x = y;
  y = z;
}
int main() {
  int firstNum = 10;
  int secondNum = 20;
  cout << "Before swap: " << "\n";
  cout << firstNum << secondNum << "\n";
  // Call the function, which will change the values of firstNum and
secondNum
  swapNums(firstNum, secondNum);
  cout << "After swap: " << "\n";
  cout << firstNum << secondNum << "\n";
  return 0;
}
```

Array passed as parameter

```
void myFunction(int myNumbers[5]) {
  for (int i = 0; i < 5; i++) {
    cout << myNumbers[i] << "\n";
  }
}

int main() {
  int myNumbers[5] = {10, 20, 30, 40, 50};
  myFunction(myNumbers);
  return 0;
}
```

Function overloading

```
int plusFunc(int x, int y) {
  return x + y;
}

double plusFunc(double x, double y) {
  return x + y;
}

int main() {
  int myNum1 = plusFunc(8, 5);
  double myNum2 = plusFunc(4.3, 6.26);
  cout << "Int: " << myNum1 << "\n";
  cout << "Double: " << myNum2;
  return 0;
}
```

String functions in C++

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

int main() {
  string str1 = "Scaler ";
  string str2 = "Topics.";

  str1.append(str2);

  cout << str1 << endl;

  return 0;
}
```

String functions in C++

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

int main() {
  string str1 = "Scaler ";
  string str2 = "Topics.";

  str1 = str1 + str2;

  cout << str1 << endl;

  return 0;
}
```

String functions in C++

Function name	Description
int length()	Used to find the string length
void swap(string& str1)	Used to swap the values of 2 strings
int compare(const string& str1)	Used to compare 2 strings
string substr(int position, int length)	Used to create a new string of a given length
string& replace(int pos, int n, string& str1)	Used to replace a portion of the string that begins at position pos and is of length of n characters
string& append(const string& str1)	Used to add new characters at the end of a string
char& at(int position)	Used to access an individual character at the given position

String functions in C++

Function name	Description
int find(string& str1, int position, int len)	Used to find the string given in the parameter
int find_first_of(string& str1, int position, int len)	Used to find the first occurrence of the given sequence
int find_last_of(string& str1, int position, int n)	Used to search the string for the last character of sequence specified
int copy(string& str1)	Used to copy the contents of a string into another

Inheritance

```
// Base class
class Vehicle {
  public:
    string brand = "Ford";
    void honk() {
      cout << "Tuut, tuut! \n" ;
    }
};
// Derived class
class Car: public Vehicle {
  public:
    string model = "Mustang";
};
int main() {
  Car myCar;
  myCar.honk();
  cout << myCar.brand + " " + myCar.model;
  return 0;
}
```

Inheritance

Base class	Private mode	Public mode	Protected mode
Private	Inaccessible	Inaccessible	Inaccessible
Protected	Private	Protected	Protected
Public	Private	Public	Protected

Friend function

```
#include <iostream>
using namespace std;
class Box
{
    private:
        int length;
    public:
        Box(): length(0) { }
        friend int printLength(Box); //friend function
};
int printLength(Box b)
{
   b.length += 10;
    return b.length;
}
int main()
{
    Box b;
    cout<<"Length of box: "<< printLength(b)<<endl;
    return 0;
}
```

```
#include <iostream>
using namespace std;
class B;          // forward declarartion.
class A
{
    int x;
    public:
    void setdata(int i)
    {
        x=i;
    }
    friend void min(A,B);         // friend function.
};
class B
{
    int y;
    public:
    void setdata(int i)
    {
        y=i;
    }
    friend void min(A,B);                    // friend function
};
void min(A a,B b)
{
    if(a.x<=b.y)
    std::cout << a.x << std::endl;
    else
    std::cout << b.y << std::endl;
}
   int main()
{
   A a;
   B b;
   a.setdata(10);
   b.setdata(20);
   min(a,b);
    return 0;
 }
```

Friend class

```
#include <iostream>
using namespace std;
class A
{
    int x =5;
    friend class B;           // friend class.
};
class B
{
  public:
    void display(A &a)
    {
        cout<<"value of x is : "<<a.x;
    }
};
int main()
{
    A a;
    B b;
    b.display(a);
    return 0;
}
```

this pointer

```
#include<iostream>
using namespace std;
/* local variable is same as a member's name */
class Test
{
private:
   int x;
public:
   void setX (int x)
   {
       // The 'this' pointer is used to retrieve the object's x
       // hidden by the local variable 'x'
       this->x = x;
   }
   void print() { cout << "x = " << x << endl; }
};
int main()
{
   Test obj;
   int x = 20;
   obj.setX(x);
   obj.print();
   return 0;
}
```

this pointer

```
/* Reference to the calling object can be returned
*/
Test& Test::func ()
{
   // Some processing
   return *this;
}
```

```
#include<iostream>
using namespace std;
class Test
{
private:
  int x;
  int y;
public:
  Test(int x = 0, int y = 0) { this->x = x; this->y = y; }
  Test &setX(int a) { x = a; return *this; }
  Test &setY(int b) { y = b; return *this; }
  void print() { cout << "x = " << x << " y = " << y << endl; }
};
int main()
{
  Test obj1(5, 5);
  // Chained function calls.  All calls modify the same object
  // as the same object is returned by reference
  obj1.setX(10).setY(20);
  obj1.print();
  return 0;
}
```

Operator overloading

- Almost all operators can be overloaded except a few. Following is the list of operators that cannot be overloaded.
- sizeof
- typeid
- Scope resolution (::)
- Class member access operators (.(dot), .* (pointer to member operator))
- Ternary or conditional (?:)

Operators that can be overloaded

- Binary Arithmetic -> +, -, *, /, %
- Unary Arithmetic -> +, -, ++, —
- Assignment -> =, +=,*=, /=,-=, %=
- Bit- wise -> & , | , << , >> , ~ , ^
- De-referencing -> (->)
- Dynamic memory allocation and De-allocation -> New, delete
- Subscript -> [ ]
- Function call -> ()
- Logical -> &, | |, !
- Relational -> >, < , = =, <=, >=

Overloading unary operator

```
// C++ program to show unary operator overloading
#include <iostream>
using namespace std;
class Distance {
public:
    int feet, inch;
       Distance(int f, int i)
    {
        this->feet = f;
        this->inch = i;
    }
   void operator-()
    {
        feet--;
        inch--;
        cout << "\nFeet & Inches(Decrement): " << feet << "" << inch;
    }
};
int main()
{
    Distance d1(8, 9);
    -d1;
    return 0;
}
```

Overloading binary operator

```
#include <iostream>
using namespace std;
class Distance {
public:
    int feet, inch;
    Distance()
    {
        this->feet = 0;
        this->inch = 0;
    }

    Distance(int f, int i)
    {
        this->feet = f;
        this->inch = i;
    }
     Distance operator+(Distance& d2) // Call by reference
    {
        Distance d3;
         d3.feet = this->feet + d2.feet;
        d3.inch = this->inch + d2.inch;
         return d3;
    }
};
```

Overloading binary operator

```
int main()
{
    // Declaring and Initializing first object
    Distance d1(8, 9);

    // Declaring and Initializing second object
    Distance d2(10, 2);

    // Declaring third object
    Distance d3;

    // Use overloaded operator
    d3 = d1 + d2;

    // Display the result
    cout << "\nTotal Feet & Inches: " << d3.feet << "'" << d3.inch;
    return 0;
}
```

Overloading binary operator

```
#include <iostream>
using namespace std;
class Distance {
public:
    int feet, inch;
    Distance()
    {
        this->feet = 0;
        this->inch = 0;
    }
   Distance(int f, int i)
    {
        this->feet = f;
        this->inch = i;
    }
     friend Distance operator+(Distance&, Distance&);
};
```

Overloading binary operator

```
Distance operator+(Distance& d1, Distance& d2)
{
    Distance d3;
    d3.feet = d1.feet + d2.feet;
    d3.inch = d1.inch + d2.inch;
    return d3;
}
 int main()
{
    Distance d1(8, 9);
    Distance d2(10, 2);
    Distance d3;
    d3 = d1 + d2;
    cout << "\nTotal Feet & Inches: " << d3.feet << "'" << d3.inch;
    return 0;
}
```

File I/O in C++

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

int main() {
  // Create and open a text file
  ofstream MyFile("filename.txt");

  // Write to the file
  MyFile << "Files can be tricky, but it is fun enough!";

  // Close the file
  MyFile.close();
}
```

File I/O in C++

```
#include <fstream>
#include <iostream>
using namespace std;
int main ()
{
    char data[100];
    ofstream myfile;
    myfile.open("E:\\message.txt");
    cout << "Writing to the file" << endl;
    cout << "Enter your name: ";
    cin.getline(data, 100);
    myfile << data << endl;
    cout << "Enter your age: "; cin >> data;
    cin.ignore();
    myfile << data << endl;
    myfile.close();
   ifstream infile;
   infile.open("E:\\message.txt");
   cout << "Reading from a file" << endl; infile >> data;
   cout << data << endl; infile >> data;
   cout << data << endl;
   infile.close();
   return 0;
}
```

File I/O in C++

```
#include <iostream>
#include <fstream>
using namespace std;
// Class to define the properties
class Employee {
public:
   string Name;
   int Employee_ID;
   int Salary;
};
int main(){
   Employee Emp_1;
   Emp_1.Name="John";
   Emp_1.Employee_ID=2121;
   Emp_1.Salary=11000;
```

```
   //Writing this data to Employee.txt
   ofstream file1;
   file1.open("Employee.txt", ios::app);
   file1.write((char*)&Emp_1,sizeof(Emp_1));
   file1.close();
   //Reading data from EMployee.txt
   ifstream file2;
   file2.open("Employee.txt",ios::in);
   file2.seekg(0);
file2.read((char*)&Emp_1,sizeof(Emp_1));
   printf("Name :%s",Emp_1.Name);
   printf("Employee ID :
%d",Emp_1.Employee_ID);
   printf("Salary :%d",Emp_1.Salary);
   file2.close();
   return 0;
}
```

Virtual Function in C++

```
#include <iostream>
using namespace std;
class base {
   void show(){
      cout << "show base
class" << endl;
   }
};
class derived : public base {
   public:
   void show(){
      cout << "show derived
class" << endl;
   }
};
```

```
int main(){
   base* bptr;
   derived d;
   bptr = &d;
   bptr->show();
}
```

Virtual Function in C++

```
#include <iostream>
using namespace std;
class base {
   public:
   virtual void print(){
      cout << "print base class" << endl;
   }
   void show(){
      cout << "show base class" << endl;
   }
};
class derived : public base {
   public:
   void print(){
      cout << "print derived class" <<
endl;
   }
   void show(){
      cout << "show derived class" <<
endl;
   }
};
```

```
int main(){
   base* bptr;
   derived d;
   bptr = &d;
   //calling virtual function
   bptr->print();
   //calling non-virtual function
   bptr->show();
}
```

Pure Virtual Function in C++

Characteristics of a pure virtual function

- **A pure virtual function is a "do nothing" function. Here "do nothing" means that it just provides the template, and derived class implements the function.**
- **It can be considered as an empty function means that the pure virtual function does not have any definition relative to the base class.**
- **Programmers need to redefine the pure virtual function in the derived class as it has no definition in the base class.**
- **A class having pure virtual function cannot be used to create direct objects of its own. It means that the class is containing any pure virtual function then we cannot create the object of that class. This type of class is known as an abstract class.**

Pure Virtual Function in C++

```
virtual void display() = 0;
                  or
virtual void display() {}
```

Pure Virtual Function in C++

```
#include <iostream>
using namespace std;
// Abstract class
class Shape
{
    public:
    virtual float calculateArea() = 0; // pure
virtual function.
};
class Square : public Shape
{
    float a;
    public:
    Square(float l)
    {
        a = l;
    }
    float calculateArea()
    {
        return a*a;
    }
};
```

```
class Circle : public Shape
{
    float r;
    public:

    Circle(float x)
    {
        r = x;
    }
    float calculateArea()
    {
        return 3.14*r*r ;
    }
};
class Rectangle : public Shape
{
    float l;
    float b;
    public:
    Rectangle(float x, float y)
    {
       l=x;
       b=y;
    }
    float calculateArea()
    {
        return l*b;
    }
};
```

Pure Virtual Function in C++

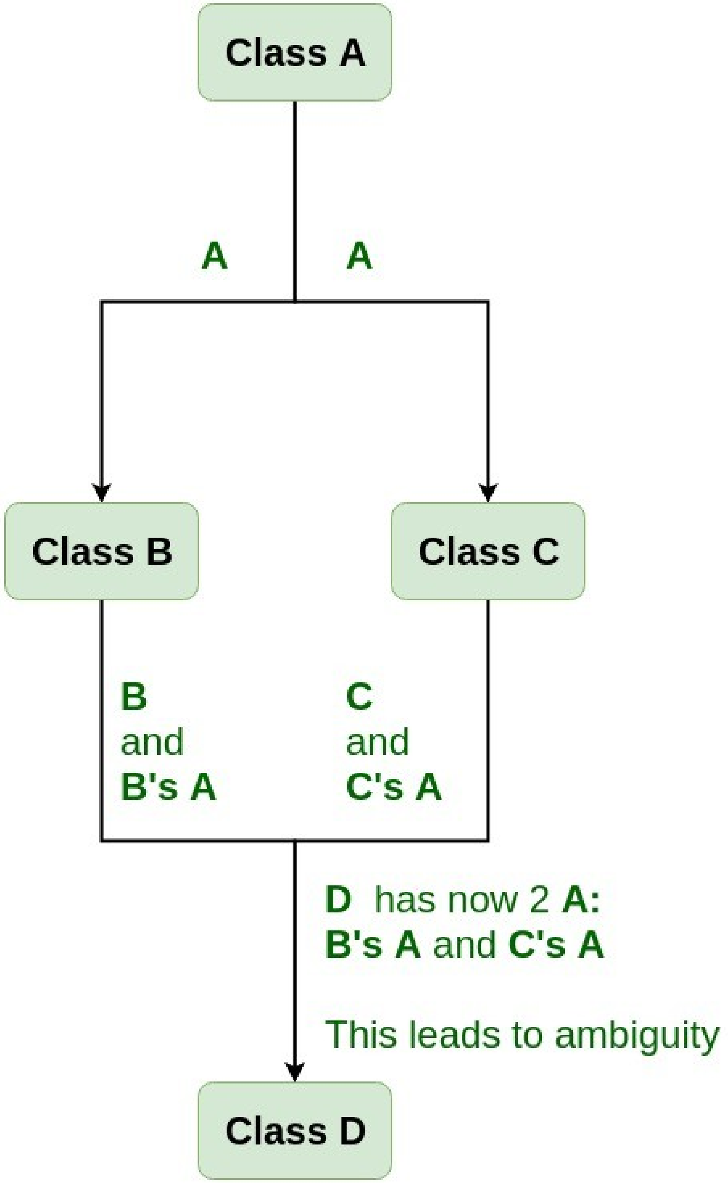
```
int main()
{

    Shape *shape;
    Square s(3.4);
    Rectangle r(5,6);
    Circle c(7.8);
    shape =&s;
    int a1 =shape->calculateArea();
    shape = &r;
    int a2 = shape->calculateArea();
    shape = &c;
    int a3 = shape->calculateArea();
    std::cout << "Area of the square is " <<a1<< std::endl;
    std::cout << "Area of the rectangle is " <<a2<< std::endl;
    std::cout << "Area of the circle is " <<a3<< std::endl;
    return 0;
}
```

Virtual Base Class in C++

- **Virtual base classes provide a way of preventing multiple “instances” of a given class appearing in an inheritance hierarchy when using multiple inheritances.**

Virtual Base Class in C++



Virtual Base Class in C++

- **As we can see from the figure that data members/function of class A are inherited twice to class D.**
- **One through class B and second through class C. When any data / function member of class A is accessed by an object of class D, ambiguity arises as to which data/function member would be called?**
- **This confuses compiler and it displays error.**

Virtual Base Class in C++

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

class A {
public:
    void show()
    {
        cout << "Hello form A \n";
    }
};

class B : public A {
};

class C : public A {
};

class D : public B, public C {
};

int main()
{
    D object;
    object.show();
}
```

Virtual Base Class in C++

Compile Errors:

```
prog.cpp: In function 'int main()':
prog.cpp:29:9: error: request for member 'show'
is ambiguous
  object.show();
         ^
prog.cpp:8:8: note: candidates are: void
A::show()
   void show()
        ^
prog.cpp:8:8: note:                 void A::show()
```

Virtual Base Class in C++

Syntax 1:

```
class B : virtual public A
{
};
```

Syntax 2:

```
class C : public virtual A
{
};
```

Virtual Base Class in C++

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

class A {
public:
    int a;
    A() // constructor
    {
        a = 10;
    }
};

class B : public virtual A {
};

class C : public virtual A {
};

class D : public B, public C {
};

int main()
{
    D object; // object creation of class d
    cout << "a = " << object.a << endl;

    return 0;
}
```

Virtual Base Class in C++

Output:

```
a = 10
```

Virtual Base Class in C++

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

class A {
public:
    void show()
    {
        cout << "Hello from A \n";
    }
};

class B : public virtual A {
};

class C : public virtual A {
};

class D : public B, public C {
};

int main()
{
    D object;
    object.show();
}
```

Virtual Base Class in C++

Output:

```
Hello from A
```

Abstract Class in C++

- A class is abstract if it has at least one pure virtual function.
- A pure virtual function (or abstract function) in C++ is a virtual function without body
- We must override that function in the derived class, otherwise the derived class will also become abstract class

Abstract Class in C++

```
#include<iostream>
using namespace std;
class Base
{
   int x;
public:
    virtual void fun() = 0;
    int getX() { return x; }
};
class Derived: public Base
{
    int y;
public:
    void fun() { cout << "fun() called"; }
};
int main(void)
{
    Derived d;
    d.fun();
    return 0;
}
```

Namespaces in C++

- Namespaces in C++ are used to organize too many classes so that it can be easy to handle the application.
- For accessing the class of a namespace, we need to use namespacename::classname.
- We can use using keyword so that we don't have to use complete name all the time.

Namespaces in C++

```
#include <iostream>
using namespace std;
namespace First {
    void sayHello() {
        cout<<"Hello First Namespace"<<endl;
    }
}
namespace Second  {
       void sayHello() {
           cout<<"Hello Second Namespace"<<endl;
       }
}
int main()
{
 First::sayHello();
 Second::sayHello();
return 0;
}
```

Namespaces in C++

```
#include <iostream>
using namespace std;
namespace First{
   void sayHello(){
      cout << "Hello First Namespace" << endl;
   }
}
namespace Second{
   void sayHello(){
      cout << "Hello Second Namespace" << endl;
   }
}
using namespace First;
int main () {
   sayHello();
   return 0;
}
```

Dynamic memory allocation in C++

```
#include <iostream>
using namespace std;
int main () {
   double* pvalue  = NULL; // Pointer initialized with null
   pvalue  = new double;   // Request memory for the variable

   *pvalue = 29494.99;     // Store value at allocated address
   cout << "Value of pvalue : " << *pvalue << endl;

   delete pvalue;         // free up the memory.

   return 0;
}
```

Dynamic memory allocation in C++

```
char* pvalue  = NULL;        // Pointer initialized with null
pvalue  = new char[20];      // Request memory for the variable
delete [] pvalue;            // Delete array pointed to by pvalue
```

Dynamic memory allocation in C++

```
#include <iostream>
using namespace std;
class Box {
  public:
    Box() {
      cout << "Constructor called!" <<endl;
    }
    ~Box() {
      cout << "Destructor called!" <<endl;
    }
};
int main() {
  Box *myBoxArray = new Box[4];
  delete [] myBoxArray; // Delete array
  return 0;
}
```

Thank you