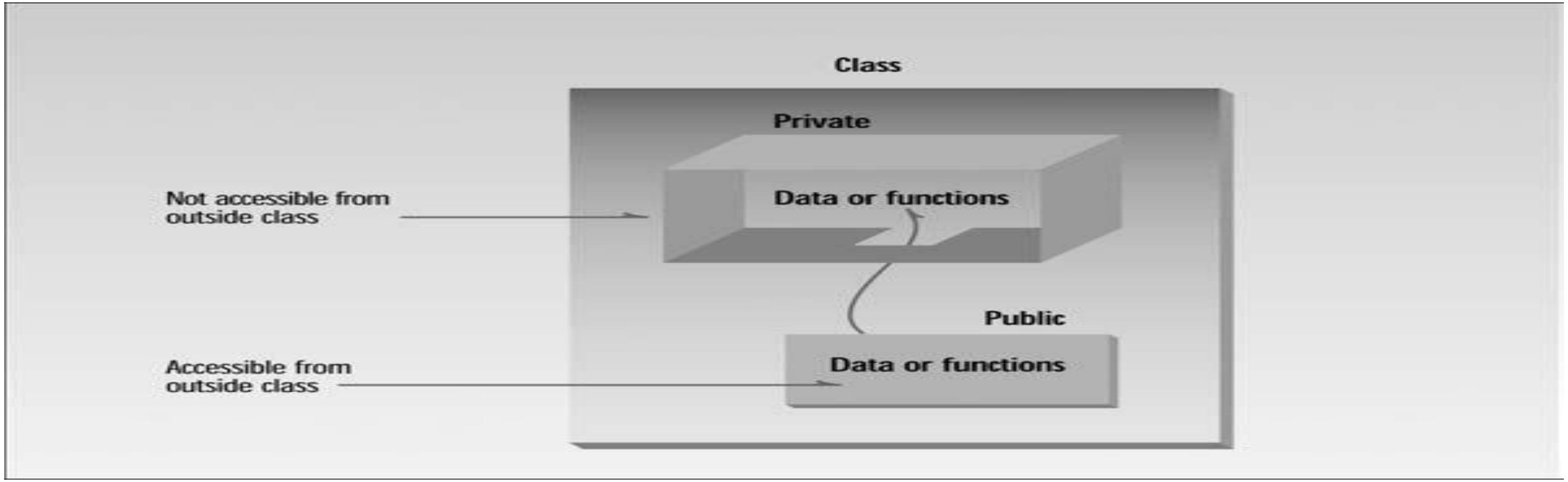
Chapter 2: Classes and Objects

- A simple class : (Recap definitions and examples)
- C++ objects as Data types
- Constructors
- Objects as function arguments
- The default copy constructor
- Returning objects from Functions
- Classes, objects and memory
- Static class data
- const and classes

Simple class-Scenario: BankAccount

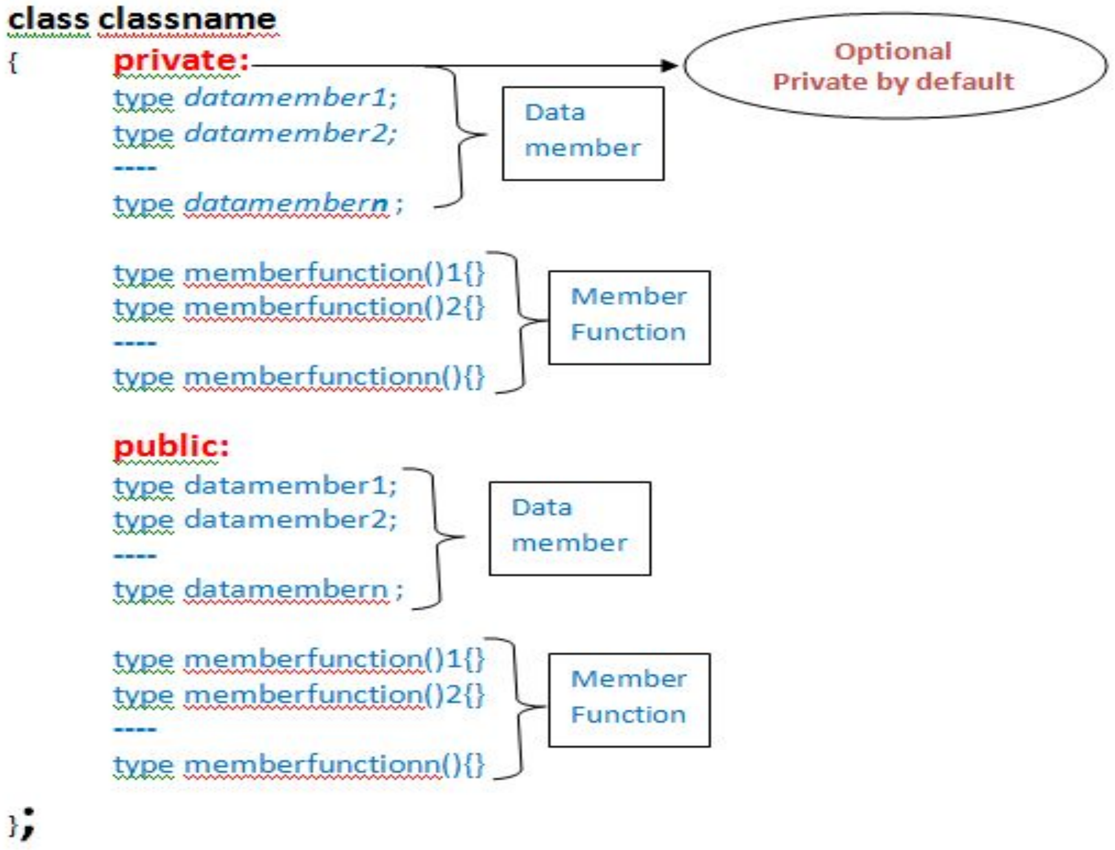
- Consider a ***bank account***, the attributes are ***accnumber*** and ***accbalance.*** Write a CPP to create a class for bank account, *instantiate* account object, *initialize* and *print* attributes of bank


Class
Private
Data or functions
Not accessible from outside class
Public
Data or functions
Accessible from outside class

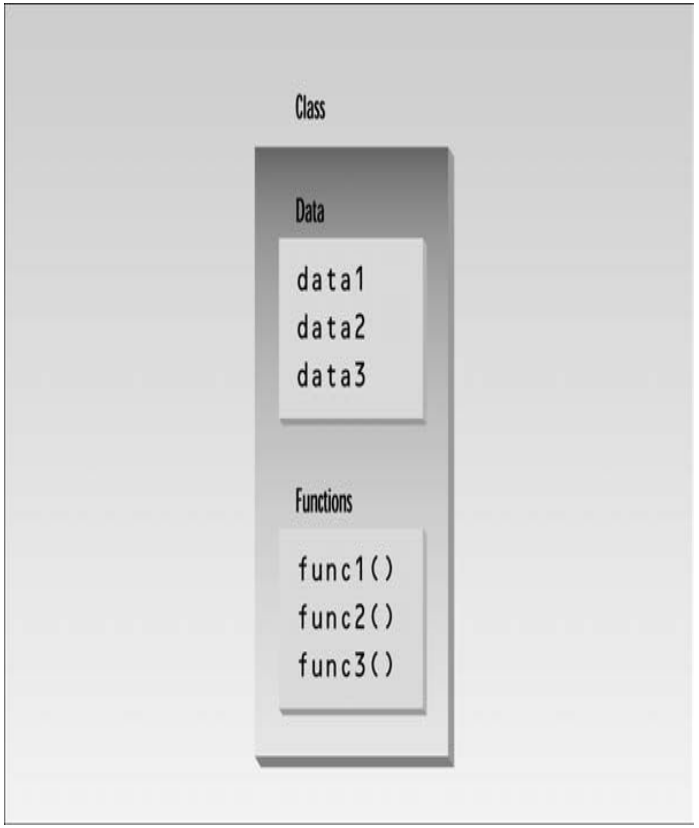
Data and functions of class

BankAccount
-accountnumber:int -accbalance
+setaccountnumber(int):void +setaccbalance(float):void +getaccountnumber():int +getaccountbalance():int

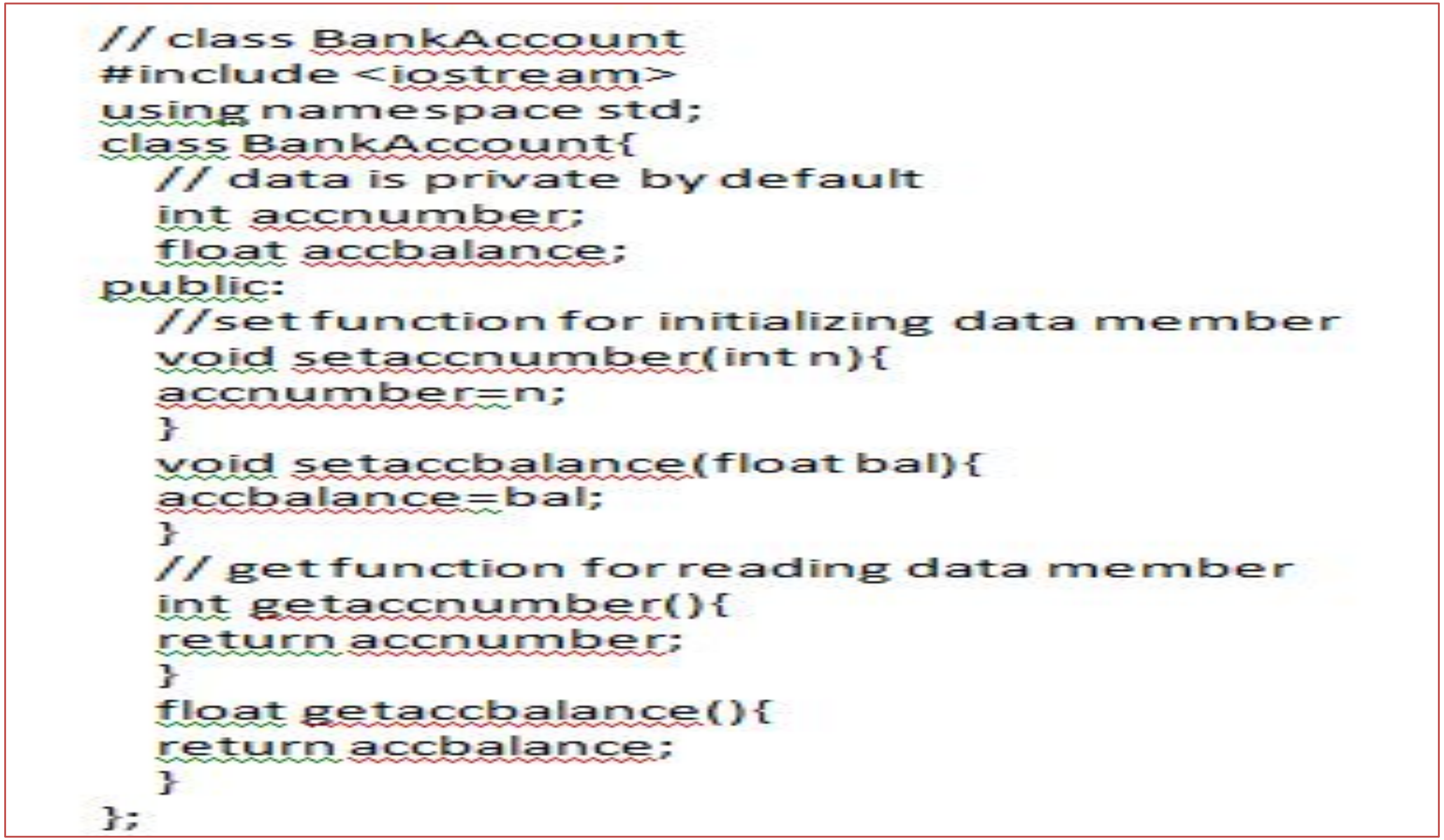
General syntax of class definition



```
class classname
{       private:
        type datamember1;
        type datamember2;
        ----
        type datamembern ;

        type memberfunction()1{}
        type memberfunction()2{}
        ----
        type memberfunctionn(){}

        public:
        type datamember1;
        type datamember2;
        ----
        type datamembern ;

        type memberfunction()1{}
        type memberfunction()2{}
        ----
        type memberfunctionn(){}
};
```

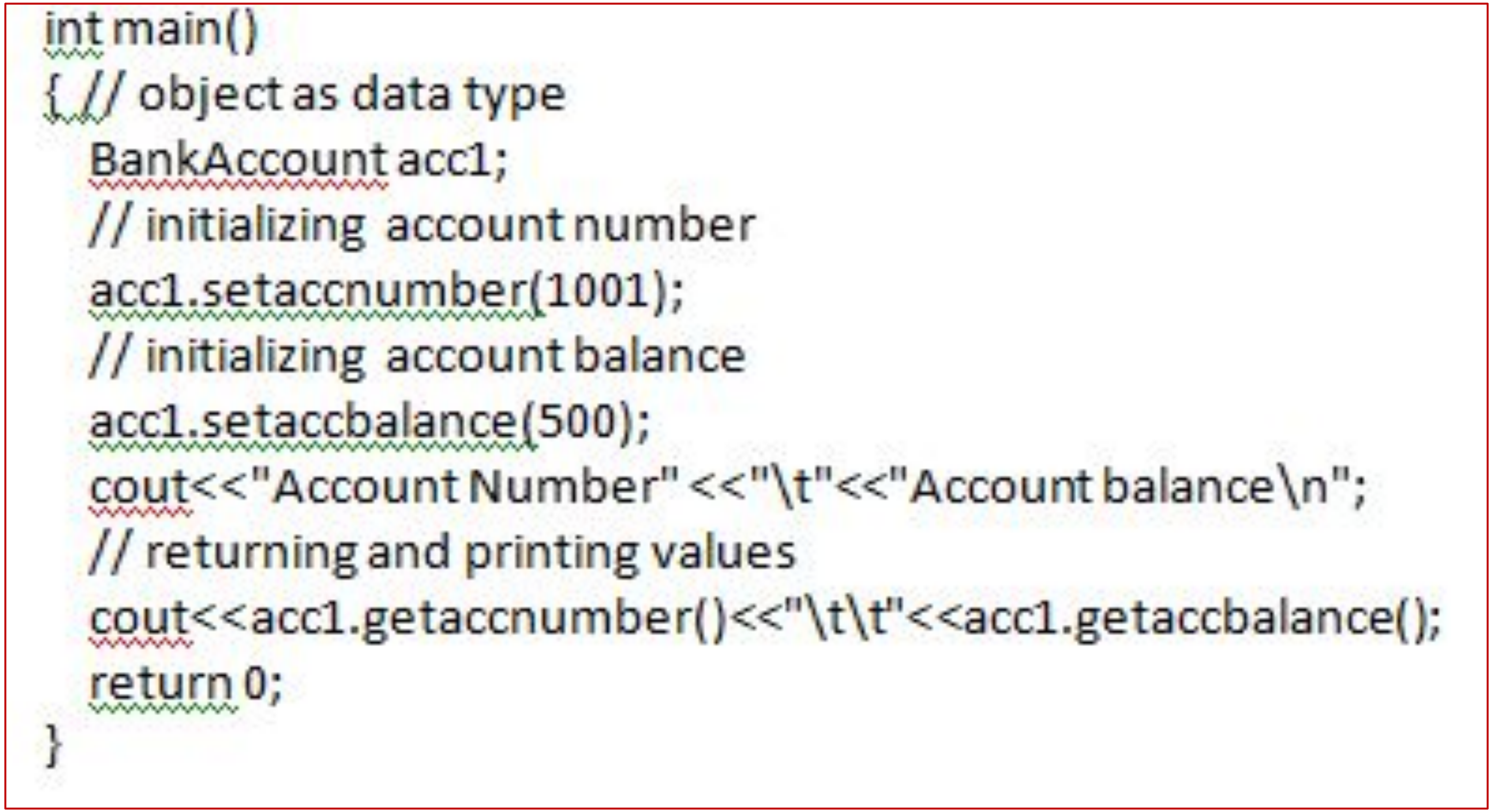


Class:BankAccount



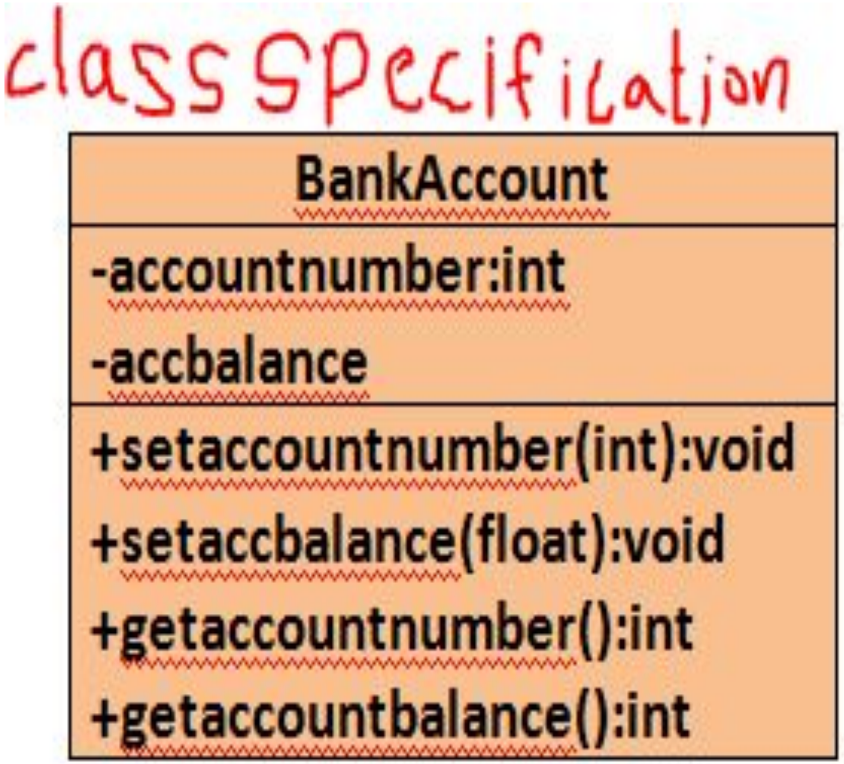
```
// class BankAccount
#include <iostream>
using namespace std;
class BankAccount{
   // data is private by default
   int accnumber;
   float accbalance;
public:
   //set function for initializing data member
   void setaccnumber(int n){
   accnumber=n;
   }
   void setaccbalance(float bal){
   accbalance=bal;
   }
   // get function for reading data member
   int getaccnumber(){
   return accnumber;
   }
   float getaccbalance(){
   return accbalance;
   }
};
```

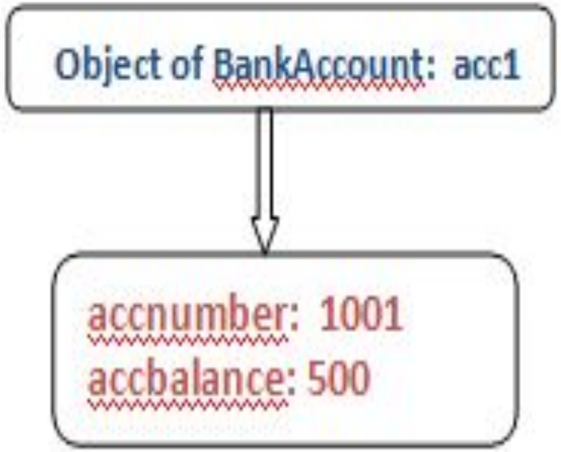
BankAccount : main function, object as a data type



```
int main()
{ // object as data type
   BankAccount acc1;
   // initializing  account number
   acc1.setaccnumber(1001);
   // initializing  account balance
   acc1.setaccbalance(500);
   cout<<"Account Number" <<"\t"<<"Account balance\n";
   // returning and printing values
   cout<<acc1.getaccnumber()<<"\t\t"<<acc1.getaccbalance();
   return 0;
}
```

Class (Object) specification and object of BankAccount

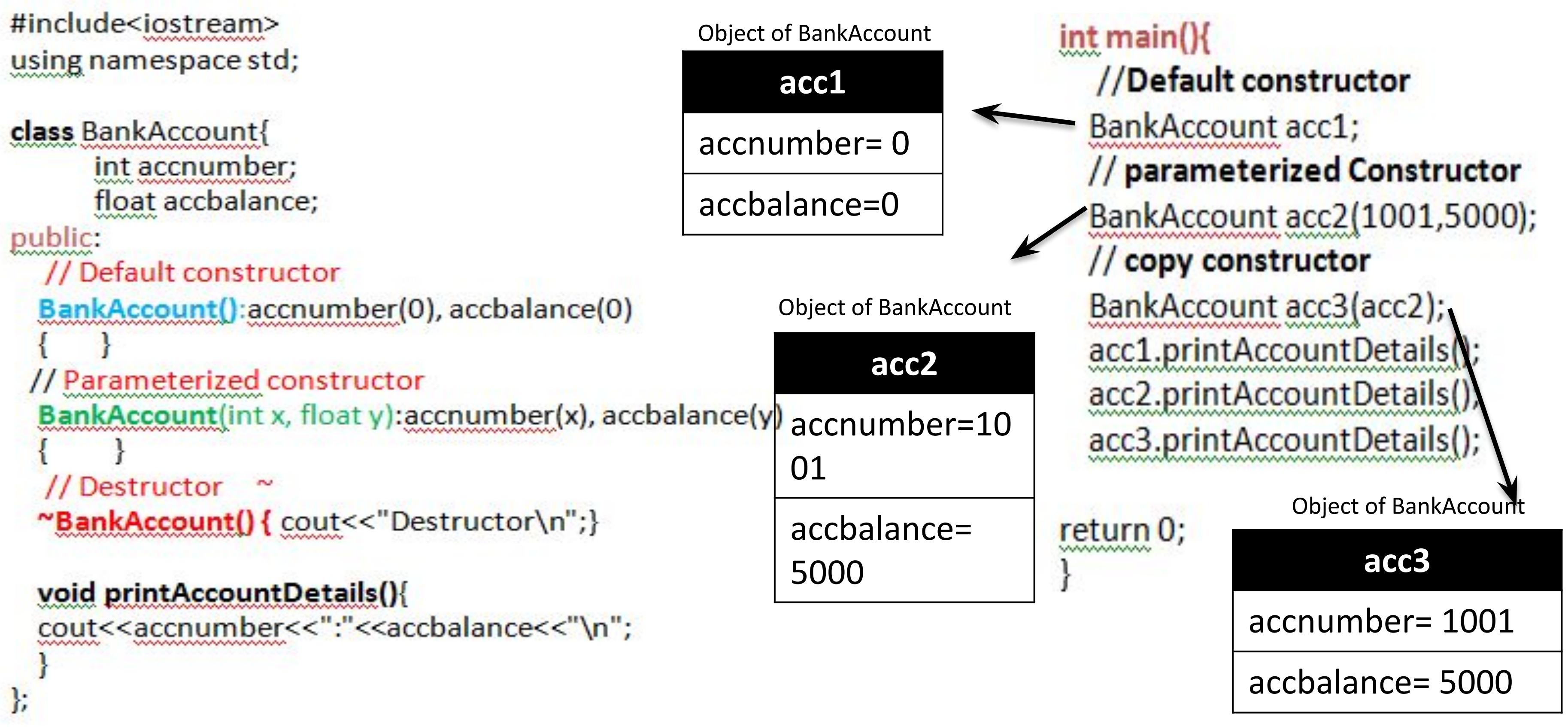




Constructor: Initializing Object data members

- The data members of object acc1 (instance of BankAccount)
  - accnumber and accbalance
  - The member functions setaccnumber(int) and setaccbalance(float) are used to initialize
  - A set function is required for every data member initialization, if forget to initialize may lead to error
  - The data members are to be initialized as and when the object is created.
  - It is implemented using a special function called ‘**constructor**’, which automatically invoked when object is created.
  - The constraint of creating constructor
    - The name of the constructor is same as class name
    - No return type for the conctructor

Constructor: Default, Parameterized, copy

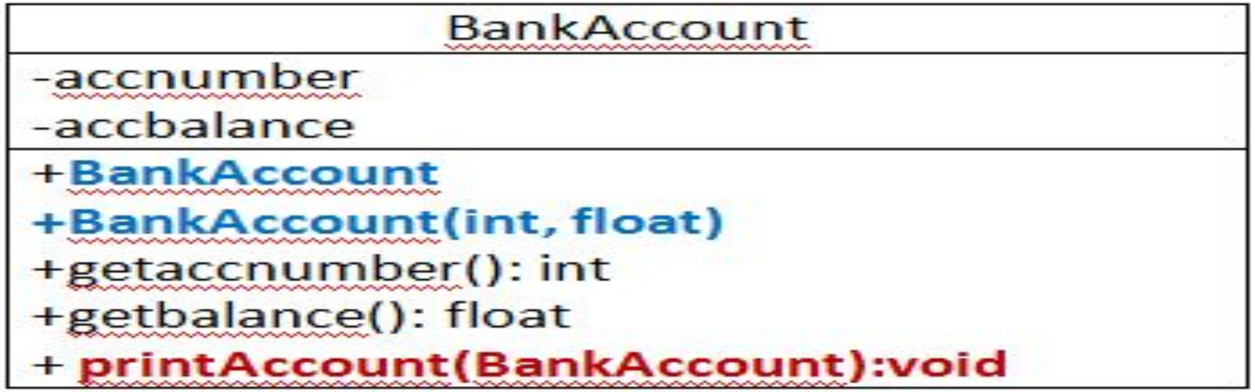


```
#include<iostream>
using namespace std;

class BankAccount{
        int accnumber;
        float accbalance;
public:
  // Default constructor
 BankAccount():accnumber(0), accbalance(0)
 {    }
// Parameterized constructor
 BankAccount(int x, float y):accnumber(x), accbalance(y)
 {     }
  // Destructor    ~
 ~BankAccount() { cout<<"Destructor\n";}

 void printAccountDetails(){
 cout<<accnumber<<":"<<accbalance<<"\n";
 }
};
```

```
int main(){
  //Default constructor
  BankAccount acc1;
  // parameterized Constructor
  BankAccount acc2(1001,5000);
  // copy constructor
  BankAccount acc3(acc2);
  acc1.printAccountDetails();
  acc2.printAccountDetails();
  acc3.printAccountDetails();

return 0;
}
```

Object as Function argument

- Consider a ***bank account***, the attributes are ***accnumber*** and ***accbalance.*** Write a CPP to create a class for bank account, *instantiate* account object, *initialize* and *print* attributes of bank account using **object as a function argument**.



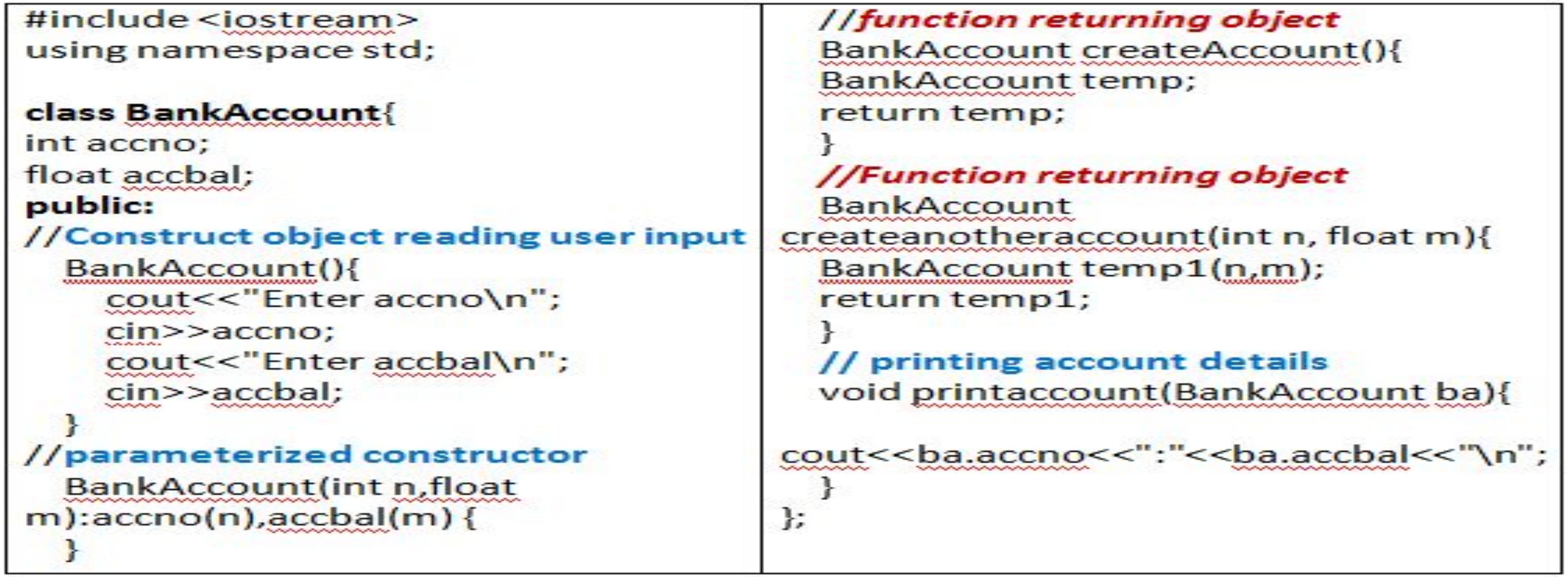
BankAccount
-accnumber -accbalance
+BankAccount +BankAccount(int, float) +getaccnumber(): int +getbalance(): float + printAccount(BankAccount):void

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

class BankAccount{
   private:
   int accnumber;
   float accbalance;
   public:
        //Default constructor
        BankAccount(): accnumber(0),accbalance(0)
        {          }
        //parameterized constructor
        BankAccount(int n, float m): accnumber(n),accbalance(m)
        {}
        int getaccnumber(){
        return accnumber;          }

       float getbalance(){
        return accbalance;          }
        // object as a argument/parameter for a function
        void printAccount(BankAccount ba){
        cout<<ba.getaccnumber()<<":"<<ba.getbalance();
        }
};
```

```
int main(){

  BankAccount acc1, acc2(1001,5000);
  cout<<acc1.getaccnumber()<<":";
  cout<<acc1.getbalance()<<"\n";

// Fun called with obj as parameter
  acc1.printAccount(acc2);
}
```

Returning objects from Functions

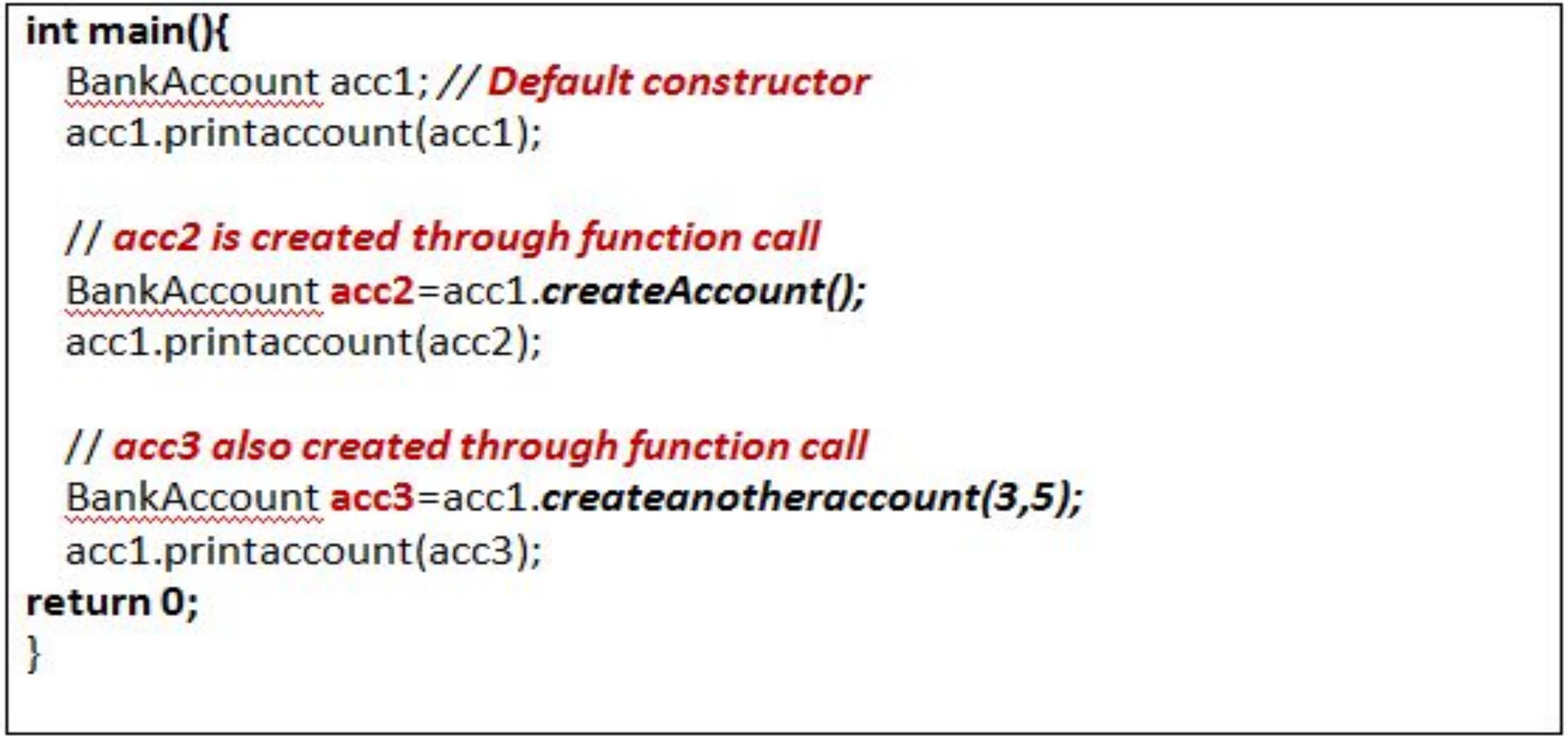
Enhance the previous example to return (create) BankAccount object using functions.



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

class BankAccount{
int accno;
float accbal;
public:
//Construct object reading user input
   BankAccount(){
      cout<<"Enter accno\n";
      cin>>accno;
      cout<<"Enter accbal\n";
      cin>>accbal;
   }
//parameterized constructor
   BankAccount(int n,float
m):accno(n),accbal(m) {
   }
```

```
   //function returning object
   BankAccount createAccount(){
   BankAccount temp;
   return temp;
   }
   //Function returning object
   BankAccount
createanotheraccount(int n, float m){
   BankAccount temp1(n,m);
   return temp1;
   }
   // printing account details
   void printaccount(BankAccount ba){

cout<<ba.accno<<":"<<ba.accbal<<"\n";
   }
};
```

Returning objects from Functions

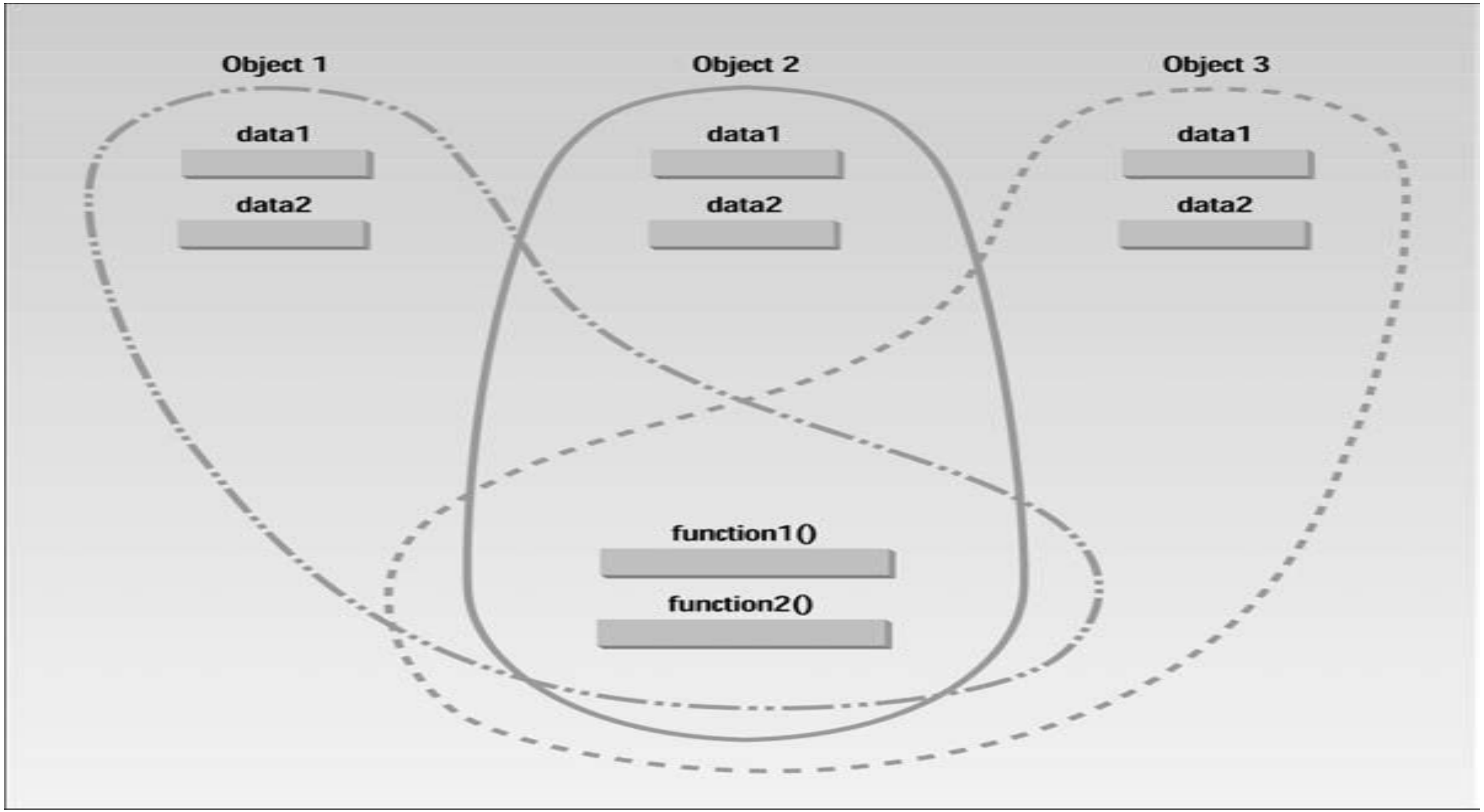


```
int main(){
   BankAccount acc1; // Default constructor
   acc1.printaccount(acc1);

   // acc2 is created through function call
   BankAccount acc2=acc1.createAccount();
   acc1.printaccount(acc2);

   // acc3 also created through function call
   BankAccount acc3=acc1.createanotheraccount(3,5);
   acc1.printaccount(acc3);
return 0;
}
```

Classes, Objects and Memory

- OO concept emphasizes that, objects are complete, self contained entities designed using the class definitions.
- The impression is, each object created from a class contains separate copies of that class’s data and member functions.
- The actual storage structure if different for object data, as all the objects of a classes use the same function.
- The member functions are created and placed in memory only once.

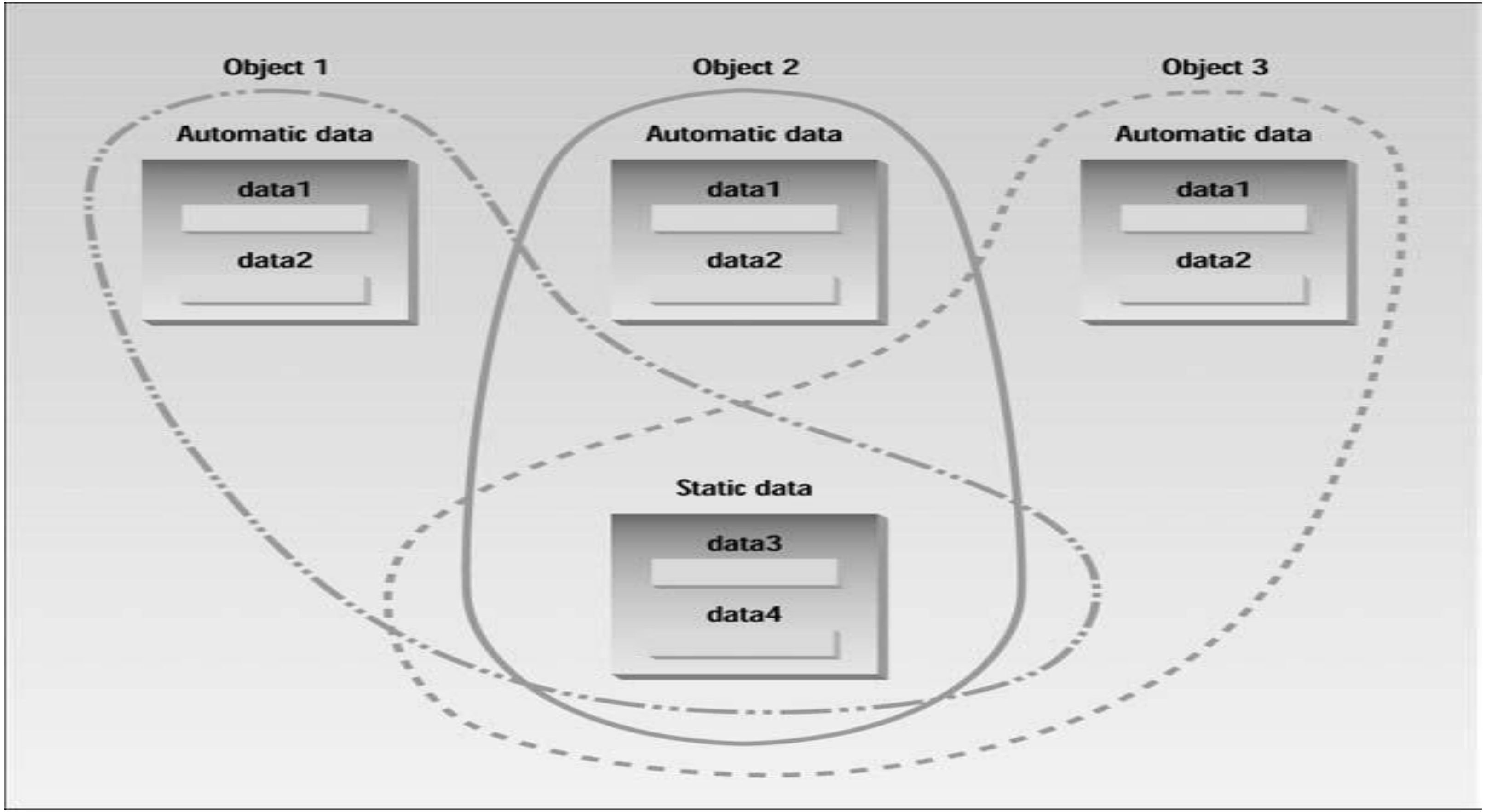
Classes, Objects and Memory



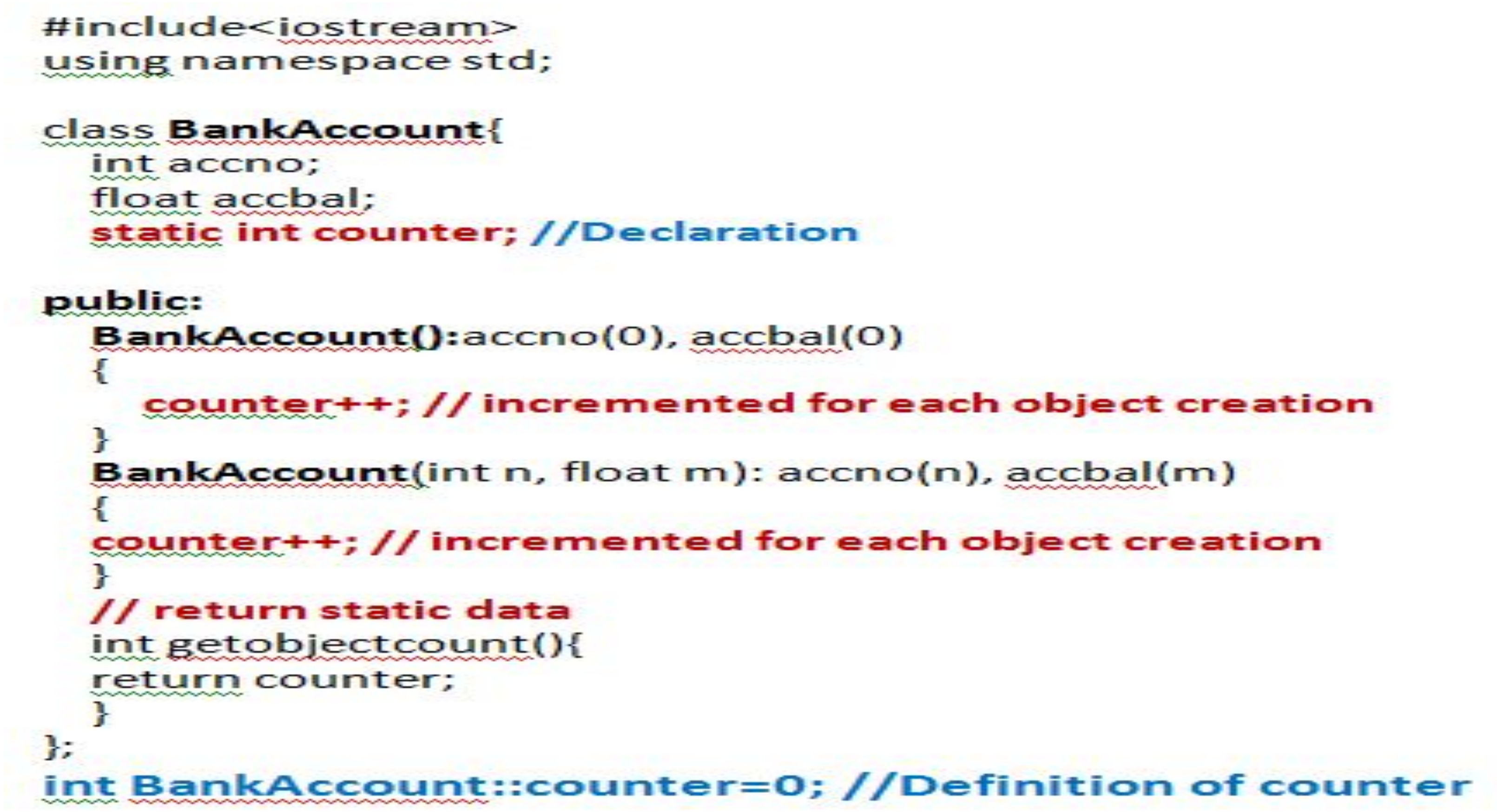
Static class data

- Each object created has its own copy of data.
- If a “static” modifier use with data member of a class, then that data is visible across objects of that class.
- The scope of the static data member is within the class, it is a class level variable.
- A static data item is useful when all objects of the same class must share a common item of information.
- Key word: “static” is used to create static data members
- Two step process to create static data member: declare and define

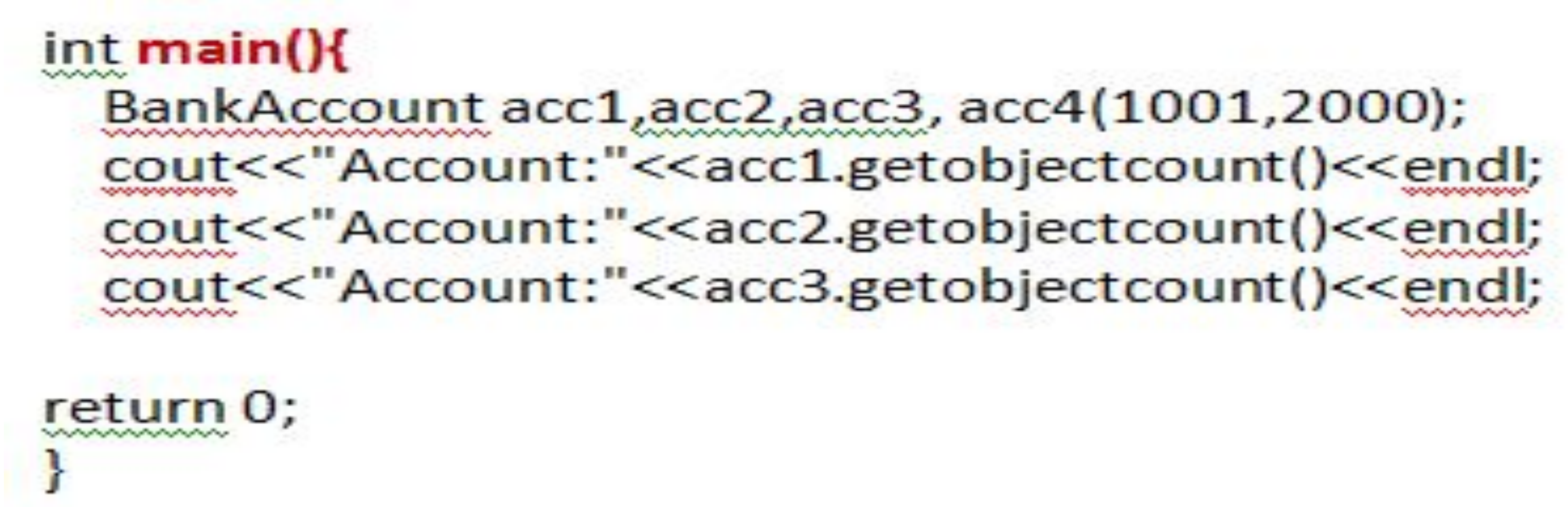
static class data



static class data: count of objects created



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

class BankAccount{
    int accno;
    float accbal;
    static int counter; //Declaration

public:
    BankAccount():accno(0), accbal(0)
    {
        counter++; // incremented for each object creation
    }
    BankAccount(int n, float m): accno(n), accbal(m)
    {
    counter++; // incremented for each object creation
    }
    // return static data
    int getobjectcount(){
    return counter;
    }
};
int BankAccount::counter=0; //Definition of counter
```

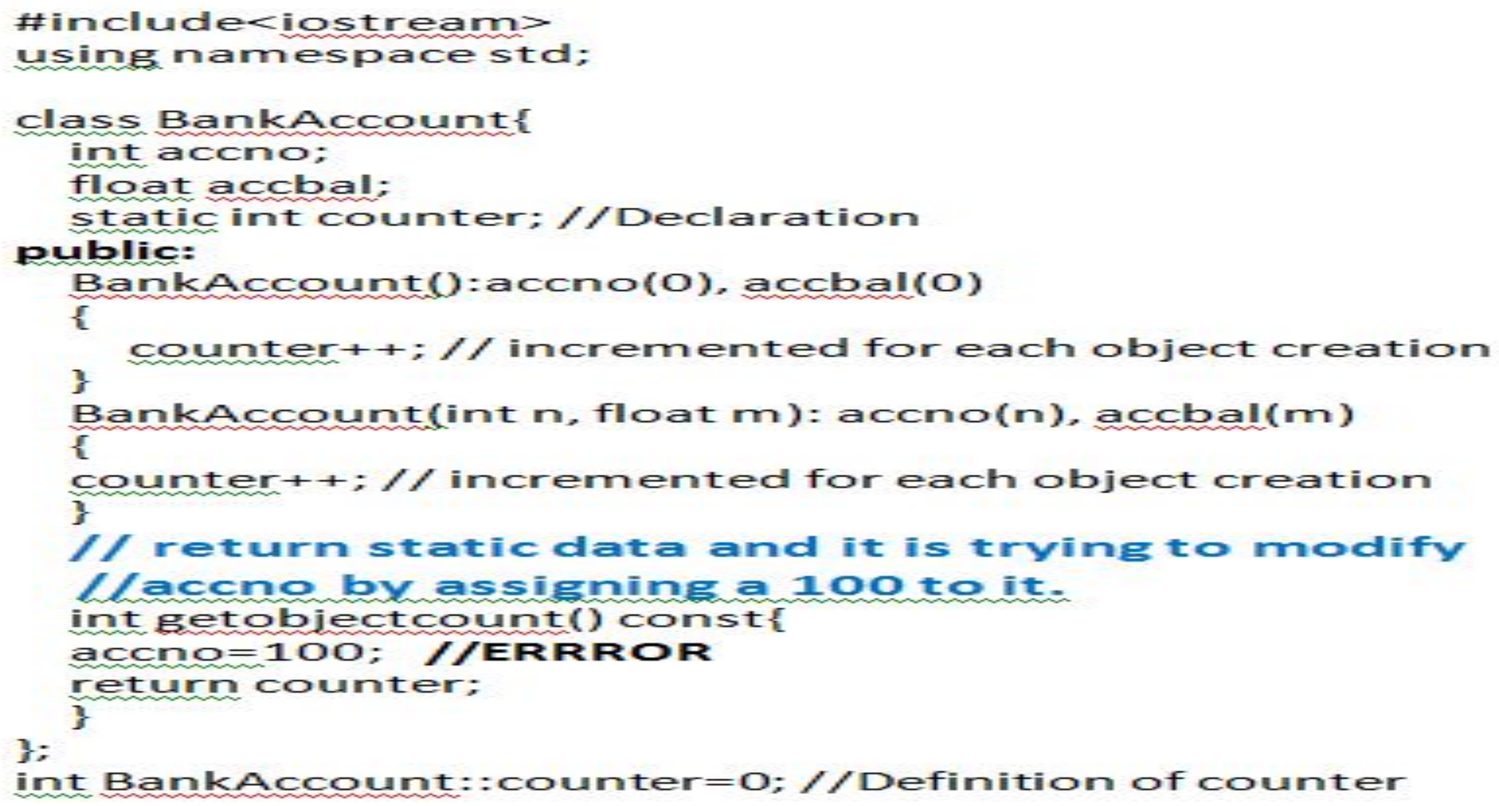


```
int main(){
   BankAccount acc1,acc2,acc3, acc4(1001,2000);
   cout<<"Account:"<<acc1.getobjectcount()<<endl;
   cout<<"Account:"<<acc2.getobjectcount()<<endl;
   cout<<"Account:"<<acc3.getobjectcount()<<endl;

return 0;
}
```

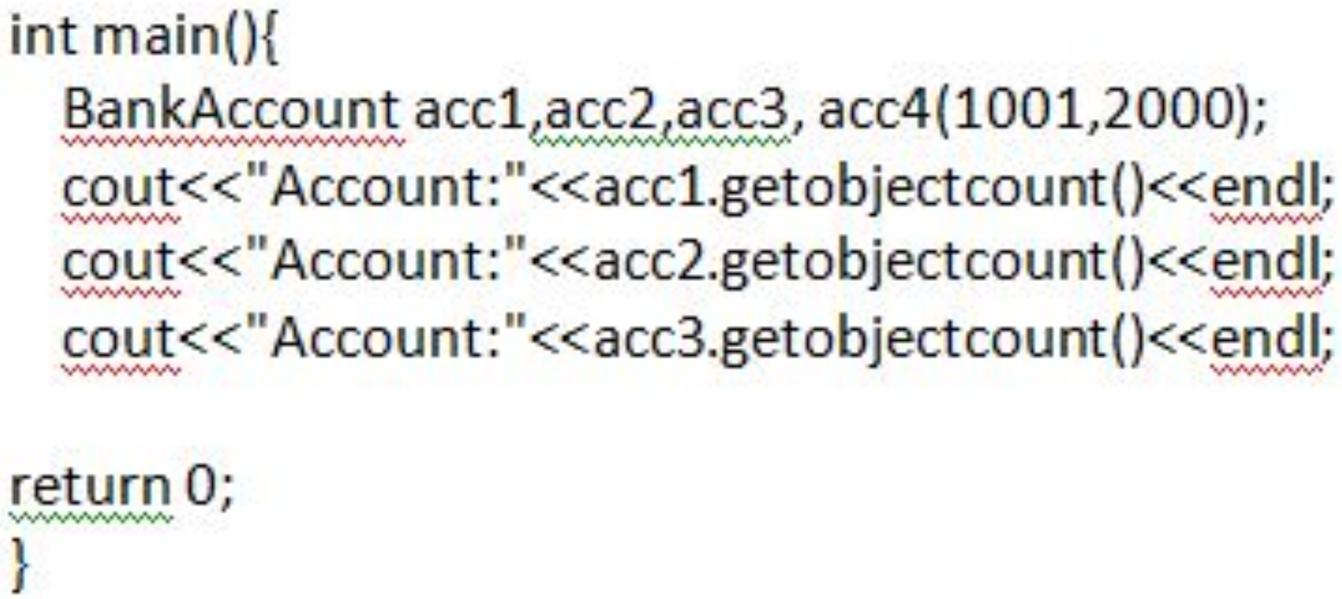
const and classes:

- const: on member function and on objects
- The const member function guarantees that it will never modify any of its class’s data member.

Const: function

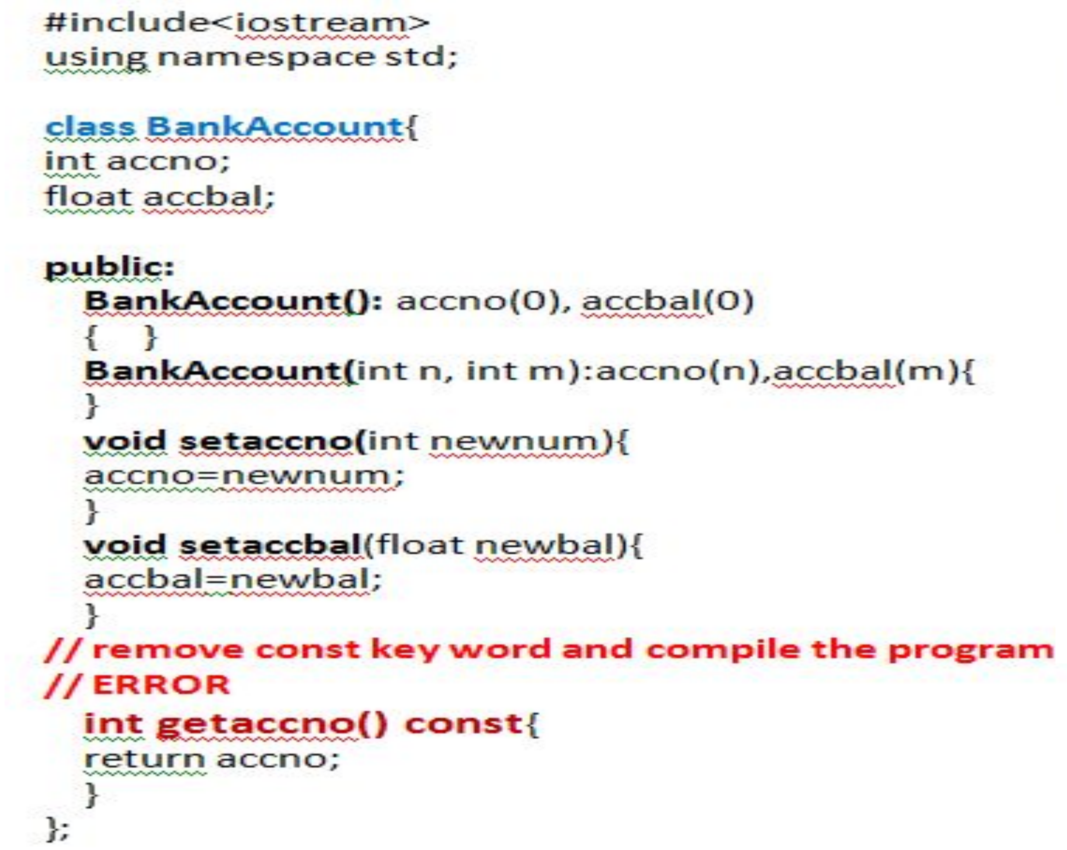


```
#include<iostream>
using namespace std;

class BankAccount{
    int accno;
    float accbal;
    static int counter; //Declaration
public:
    BankAccount():accno(0), accbal(0)
    {
        counter++; // incremented for each object creation
    }
    BankAccount(int n, float m): accno(n), accbal(m)
    {
    counter++; // incremented for each object creation
    }
    // return static data and it is trying to modify
    //accno by assigning a 100 to it.
    int getobjectcount() const{
    accno=100;   //ERRROR
    return counter;
    }
};
int BankAccount::counter=0; //Definition of counter
```

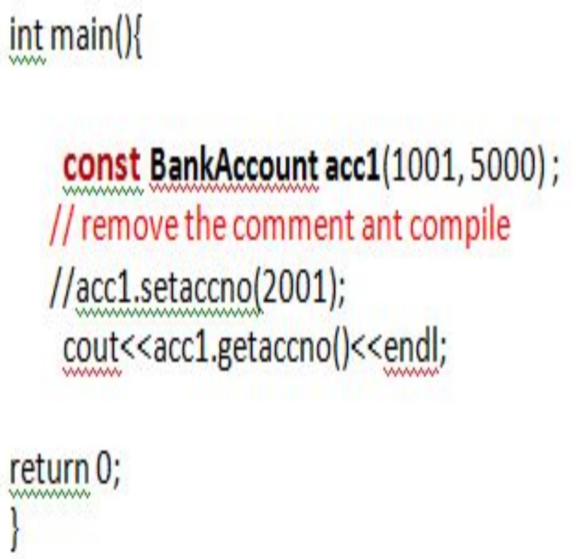


```
int main(){
    BankAccount acc1,acc2,acc3, acc4(1001,2000);
    cout<<"Account:"<<acc1.getobjectcount()<<endl;
    cout<<"Account:"<<acc2.getobjectcount()<<endl;
    cout<<"Account:"<<acc3.getobjectcount()<<endl;

return 0;
}
```

Const: objects

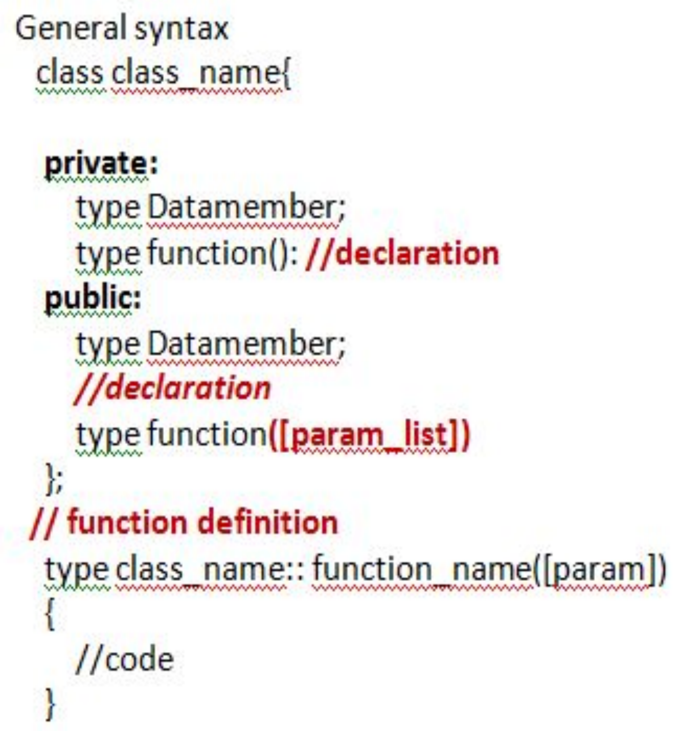


```
#include<iostream>
using namespace std;

class BankAccount{
int accno;
float accbal;

public:
   BankAccount(): accno(0), accbal(0)
   {    }
   BankAccount(int n, int m):accno(n),accbal(m){
   }
   void setaccno(int newnum){
   accno=newnum;
   }
   void setaccbal(float newbal){
   accbal=newbal;
   }
// remove const key word and compile the program
// ERROR
   int getaccno() const{
   return accno;
   }
};
```

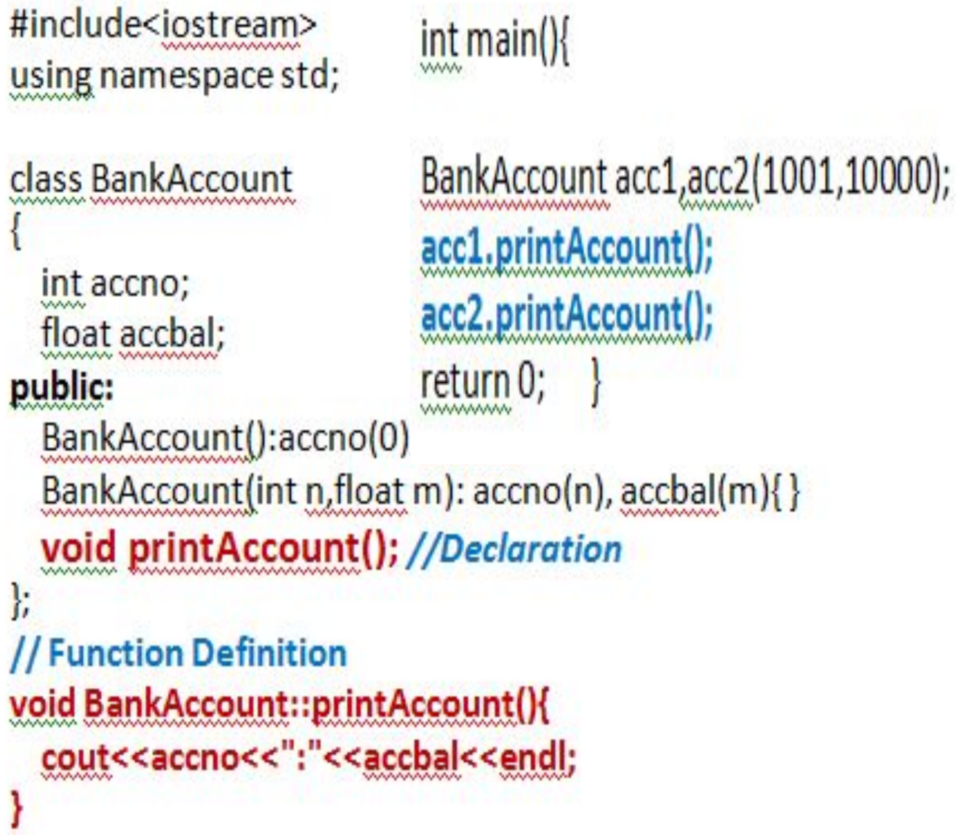


```
int main(){


   const BankAccount acc1(1001, 5000) ;
  // remove the comment ant compile
  //acc1.setaccno(2001);
   cout<<acc1.getaccno()<<endl;


return 0;
}
```

Member function defined out side the class



```
General syntax
  class class_name{

   private:
      type Datamember;
      type function(): //declaration
   public:
      type Datamember;
      //declaration
      type function([param_list])
   };
 // function definition
   type class_name:: function_name([param])
   {
      //code
   }
```



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

class BankAccount
{
   int accno;
   float accbal;
public:
   BankAccount():accno(0)
   BankAccount(int n,float m): accno(n), accbal(m){ }
   void printAccount(); //Declaration
};
// Function Definition
void BankAccount::printAccount(){
   cout<<accno<<":"<<accbal<<endl;
}
```

```
int main(){

BankAccount acc1,acc2(1001,10000);
acc1.printAccount();
acc2.printAccount();
return 0;    }
```

Array

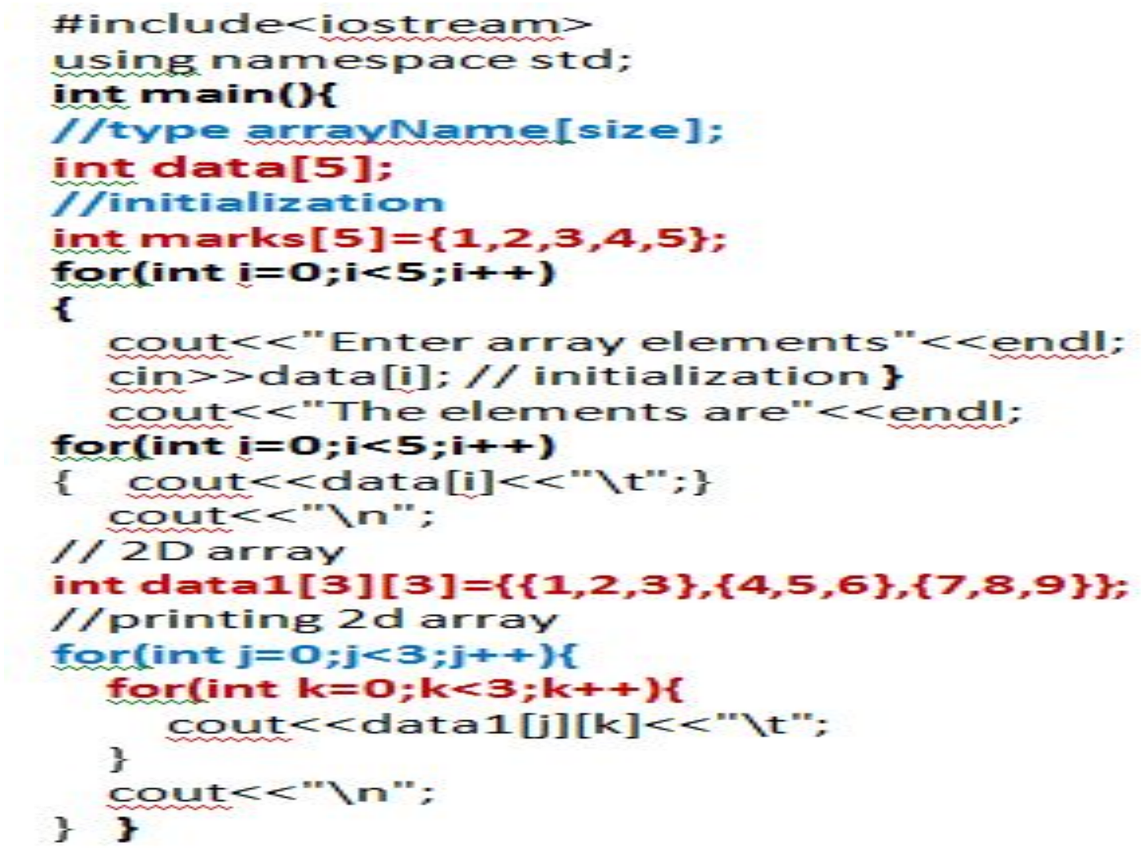
- C++ uses c-style array: declaration and definition
- Two types: 1D and 2D

General syntax:

type array_name[size];

Type: primitive type/ user defined types (objects)

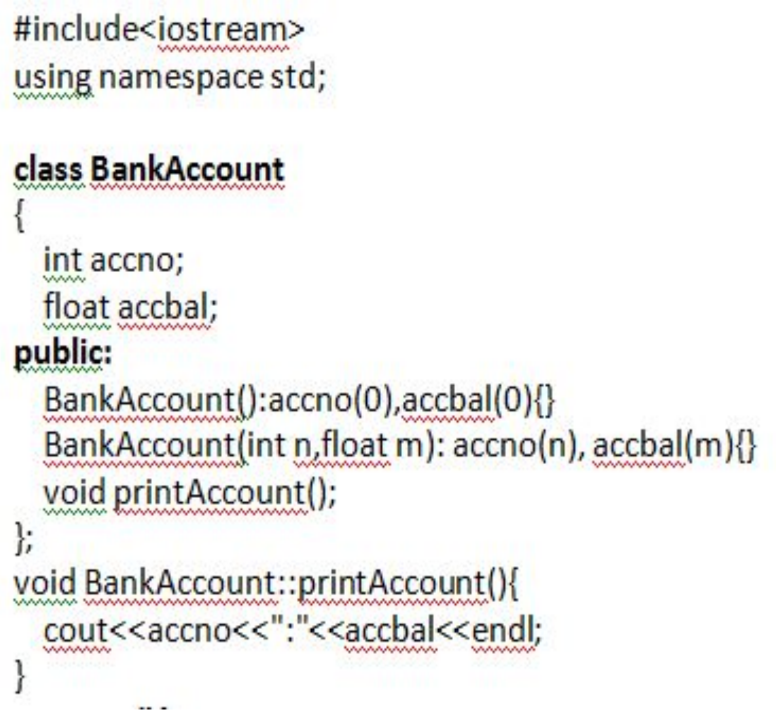
Array syntax and example



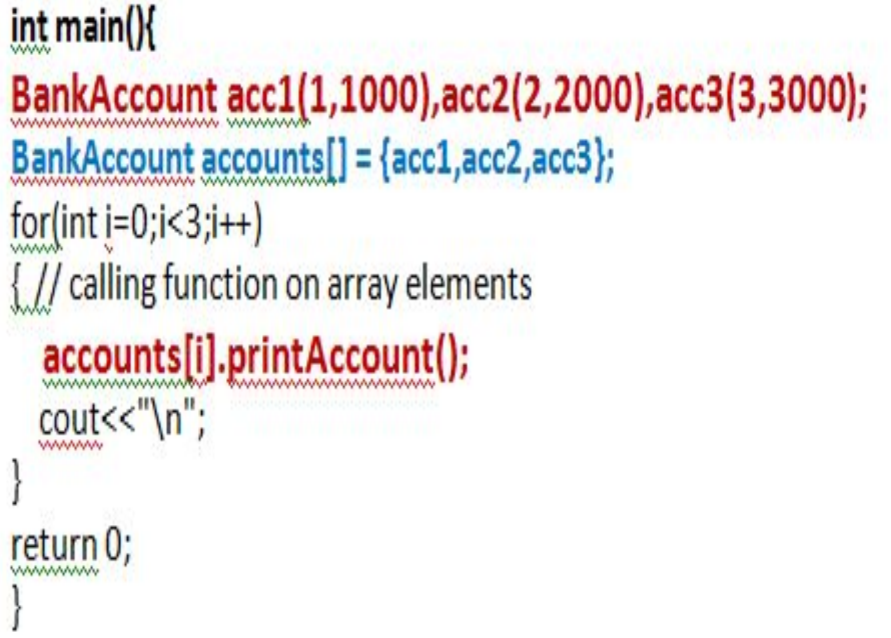
```
#include<iostream>
using namespace std;
int main(){
//type arrayName[size];
int data[5];
//initialization
int marks[5]={1,2,3,4,5};
for(int i=0;i<5;i++)
{
   cout<<"Enter array elements"<<endl;
   cin>>data[i]; // initialization }
   cout<<"The elements are"<<endl;
for(int i=0;i<5;i++)
{   cout<<data[i]<<"\t";}
   cout<<"\n";
// 2D array
int data1[3][3]={{1,2,3},{4,5,6},{7,8,9}};
//printing 2d array
for(int j=0;j<3;j++){
   for(int k=0;k<3;k++){
      cout<<data1[j][k]<<"\t";
   }
   cout<<"\n";
}  }
```

Array of objects

- Enhance the previous BankAccount class and use array of objects and call methods

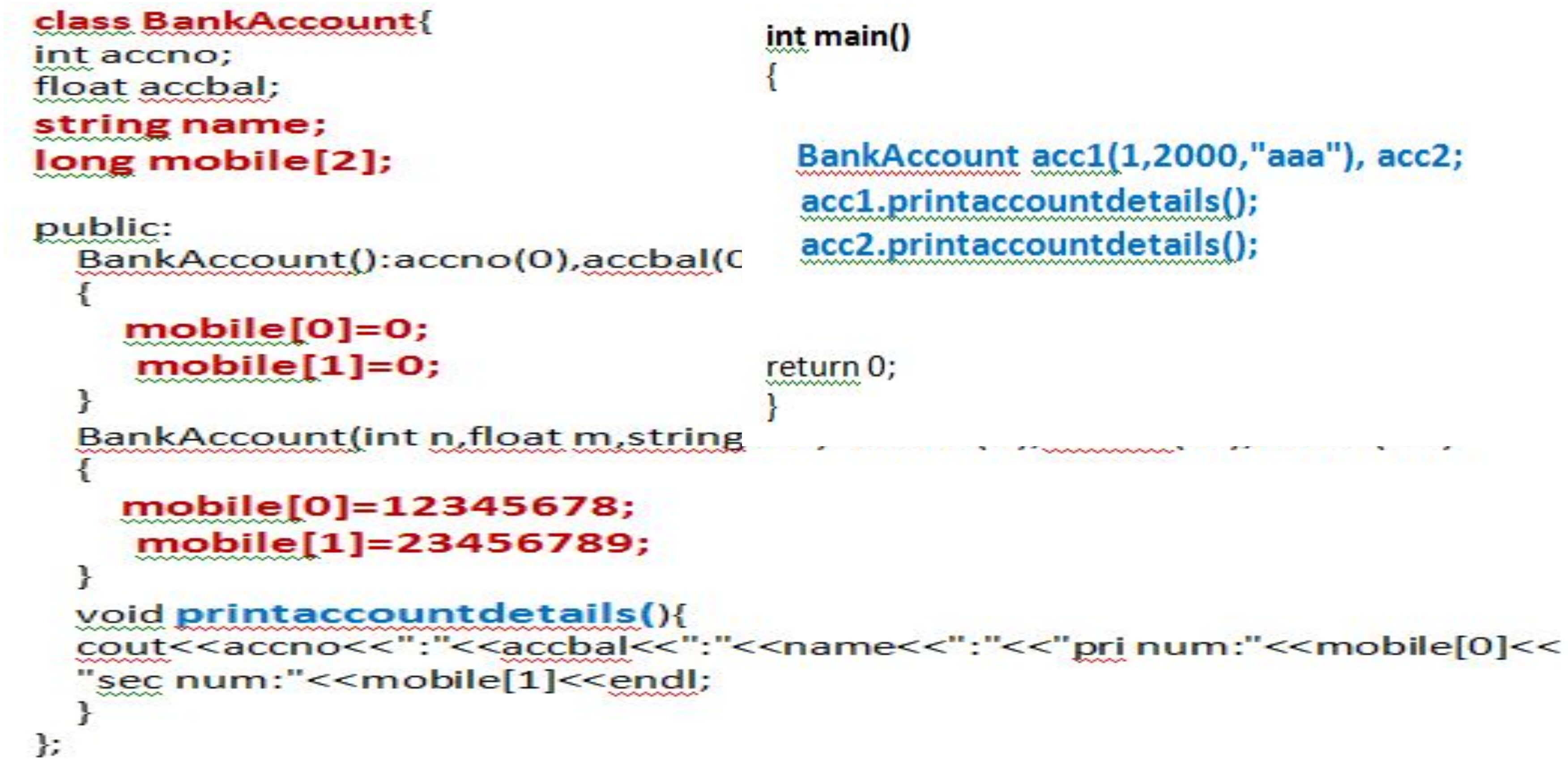


```
#include<iostream>
using namespace std;

class BankAccount
{
   int accno;
   float accbal;
public:
   BankAccount():accno(0),accbal(0){}
   BankAccount(int n,float m): accno(n), accbal(m){}
   void printAccount();
};
void BankAccount::printAccount(){
   cout<<accno<<":"<<accbal<<endl;
}
```

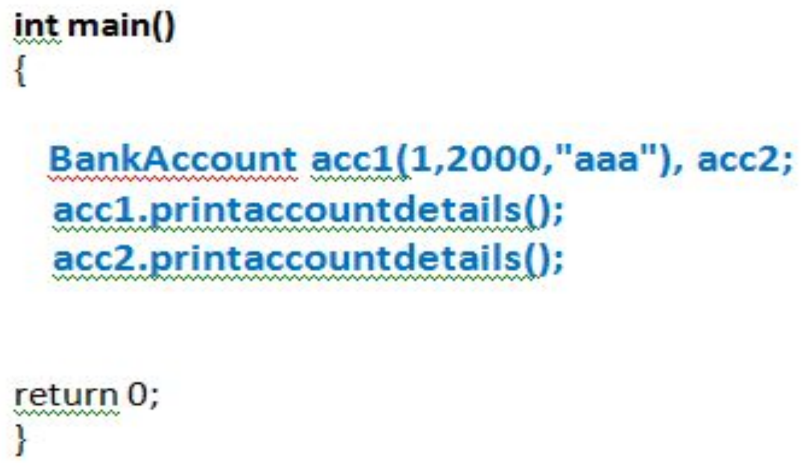


```
int main(){
BankAccount acc1(1,1000),acc2(2,2000),acc3(3,3000);
BankAccount accounts[] = {acc1,acc2,acc3};
for(int i=0;i<3;i++)
{ // calling function on array elements
   accounts[i].printAccount();
   cout<<"\n";
}
return 0;
}
```

Redefine the BankAccount class to include the additional attributes: name and mobile The account holder has two mobiles.



```
class BankAccount{
int accno;
float accbal;
string name;
long mobile[2];

public:
   BankAccount():accno(0),accbal(C
   {
      mobile[0]=0;
       mobile[1]=0;
   }
   BankAccount(int n,float m,string
   {
      mobile[0]=12345678;
       mobile[1]=23456789;
   }
   void printaccountdetails(){
   cout<<accno<<":"<<accbal<<":"<<name<<":"<<"pri num:"<<mobile[0]<<
   "sec num:"<<mobile[1]<<endl;
   }
};
```

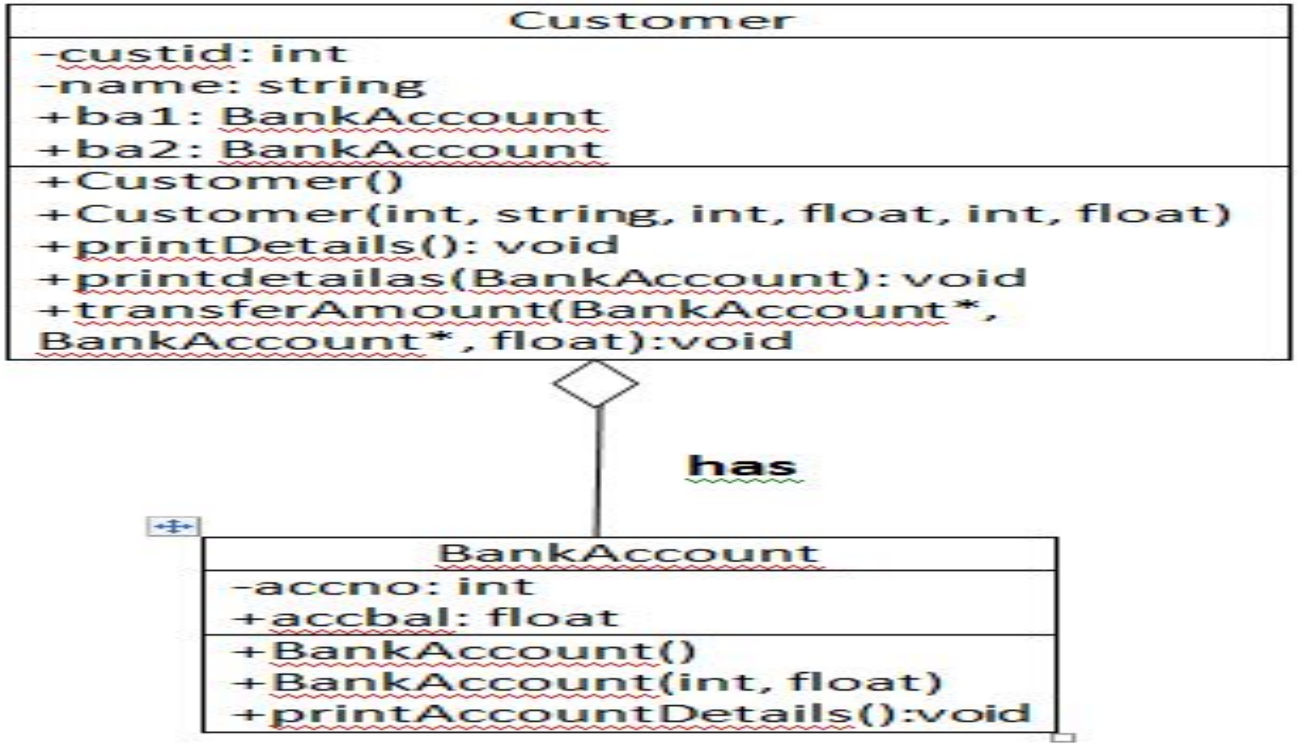


```
int main()
{

   BankAccount acc1(1,2000,"aaa"), acc2;
   acc1.printaccountdetails();
   acc2.printaccountdetails();


return 0;
}
```

Scenario:

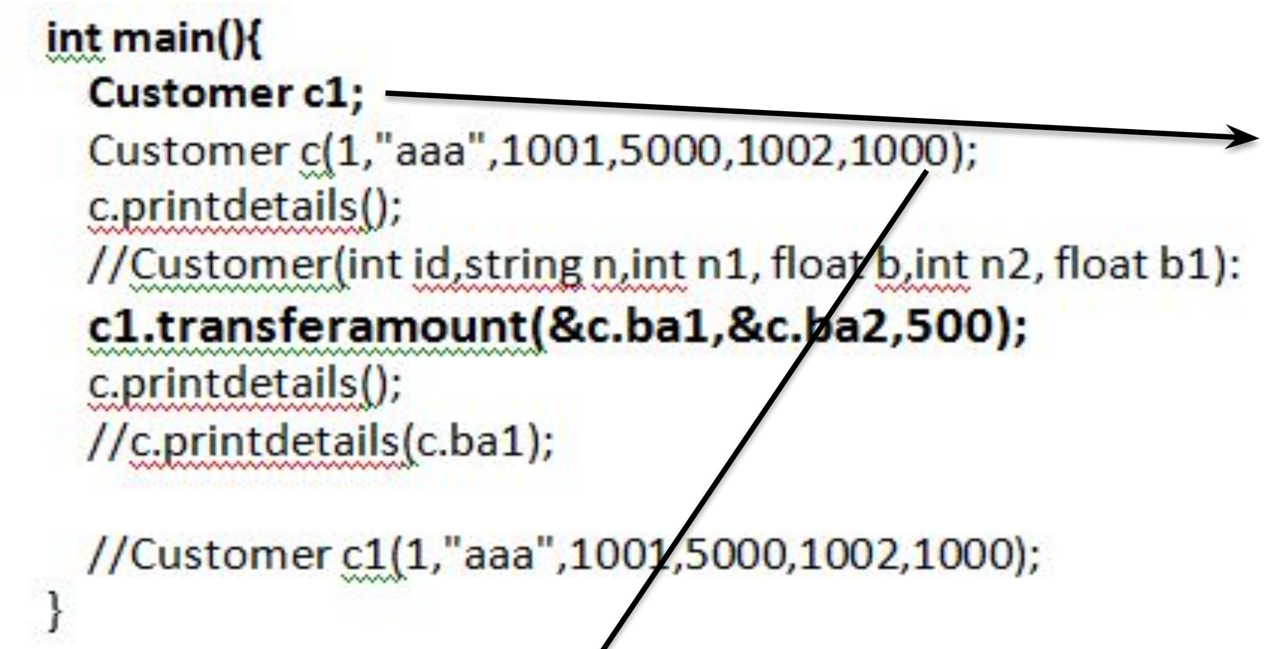
Consider a customer has bank accounts, they are of type savings accounts. He can credit, debit money to his accounts and transfer amount from one account to another. Both the accounts shall have minimum balance of Rs. 500. Write a cpp program to simulate the above scenario.

Relationship: Customer has BankAccount



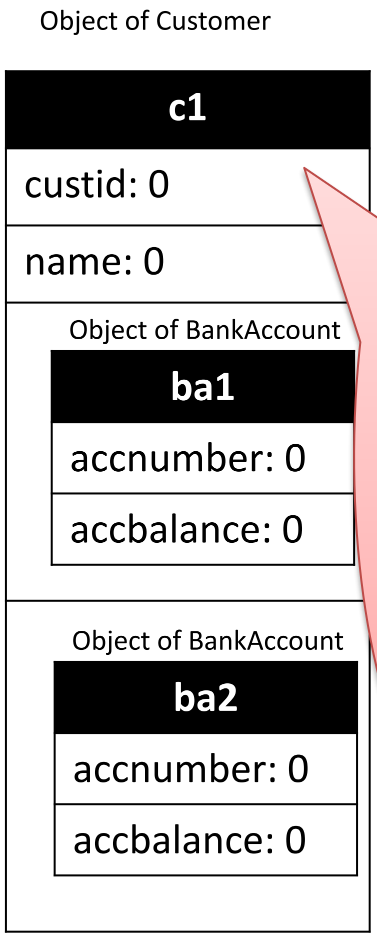
```
#include<iostream>
using namespace std;
class BankAccount{
int accno;
//float accbal;
public:
    float accbal;
    BankAccount():accno(0),accbal(0)
    {    }
    BankAccount(int n,float b):accno(n),accbal(b)
    {    }
    int getaccno(){
    return accno;
    }
    float getbal(){
    return accbal;
    }
    void printAccountDetails(){
    cout<<accno<<":"<<accbal<<"\n";
    }
};
```

```
class Customer{
int custid;
string name;
//BankAccount ba1,ba2;

public:
   BankAccount ba1,ba2;
   Customer():custid(0),name(""), ba1()
   {       }
   Customer(int id,string n,int n1, float b,int n2, float b1):
      custid(id),name(n),ba1(n1,b),ba2(n2,b1)
   {          }
   void transferamount(BankAccount* b1, BankAccount* b2,float amt){
   b1->accbal=b1->accbal-amt;
   b2->accbal+=amt;
   //cout<<b1.accbal;   }
   void printdetails(){
cout<<custid<<":"<<name<<":"<<ba1.getaccno()<<":"<<ba1.getbal()<<":"<<ba
2.getaccno()<<":"<<ba2.getbal()<<"\n";
   }
   void printdetails(BankAccount b){
   cout<<custid<<":"<<name<<":"<<b.getaccno()<<":"<<b.getbal();
   }
};
```



```
int main(){
    Customer c1;
    Customer c(1,"aaa",1001,5000,1002,1000);
    c.printdetails();
    //Customer(int id,string n,int n1, float b,int n2, float b1):
    c1.transferamount(&c.ba1,&c.ba2,500);
    c.printdetails();
    //c.printdetails(c.ba1);

    //Customer c1(1,"aaa",1001,5000,1002,1000);
}
```

Draw the object visualization for the customer object created in this line



Note: This is **not** Class Diagram

Summary:

1. Implemented classes (private, public, data, constructor, functions)
2. Implemented object as function argument and return values
3. Implemented static data members , const: functions and objects

Scenario:

- Consider a Debit Card issued to a customer by a Bank. The debit card is of type ‘Rupay’. The customer can swipe the card for paying his purchases and he is allowed to change the pin. Identify the relevant attributes and functions the customer can perform using debit card. Draw a class diagram using proper UML notations.

Scenario:

- Consider a student of 4th semester, he register for multiple courses. The student is allowed to change his address and email. Identify relevant attributes for student object and functions performed by him. Draw a class diagram using UML notations. Print the student details before and after modifications.

Copy Constructor

- A copy constructor is a member function which initializes an object using another object of the same class. A copy constructor has the following general function prototype:
- ClassName (const ClassName &oldObject);

Copy Constructor

- In C++, a Copy Constructor may be called in following cases:
  1. When an object of the class is returned by value.
  2. When an object of the class is passed (to a function) by value as an argument.
  3. When an object is constructed based on another object of the same class.
  4. When the compiler generates a temporary object.

Copy Constructor

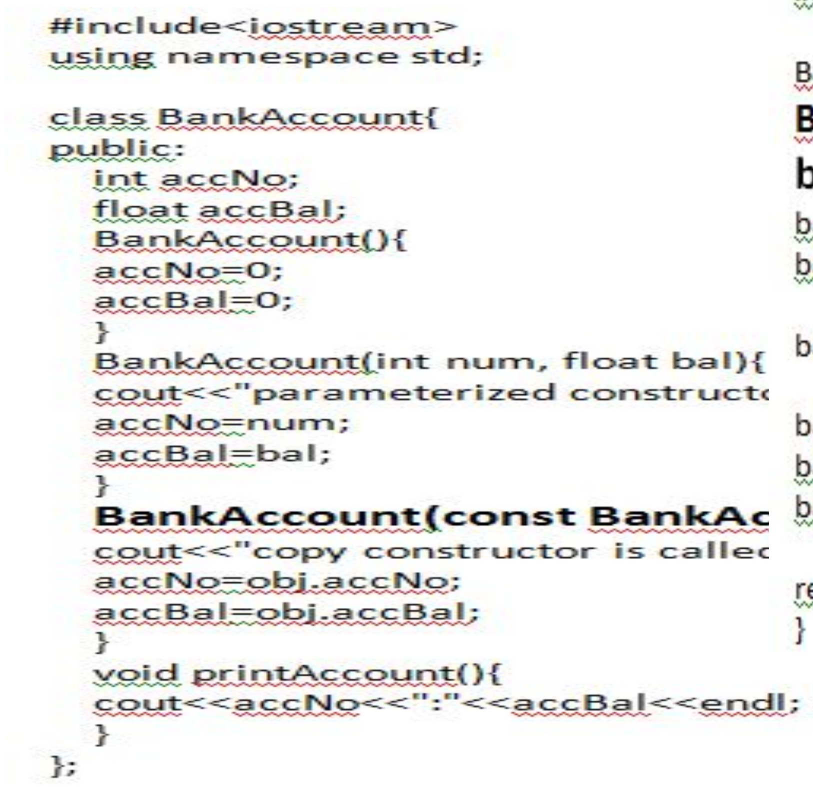
- **Copy constructor vs Assignment Operator**
  Which of the following two statements call copy constructor and which one calls assignment operator?

```
BankAccount      ba1, ba2(1001,5000);
BankAccount ba3=ba1; // calls copy constructor
ba2=ba1;        // calls assignment operator
```

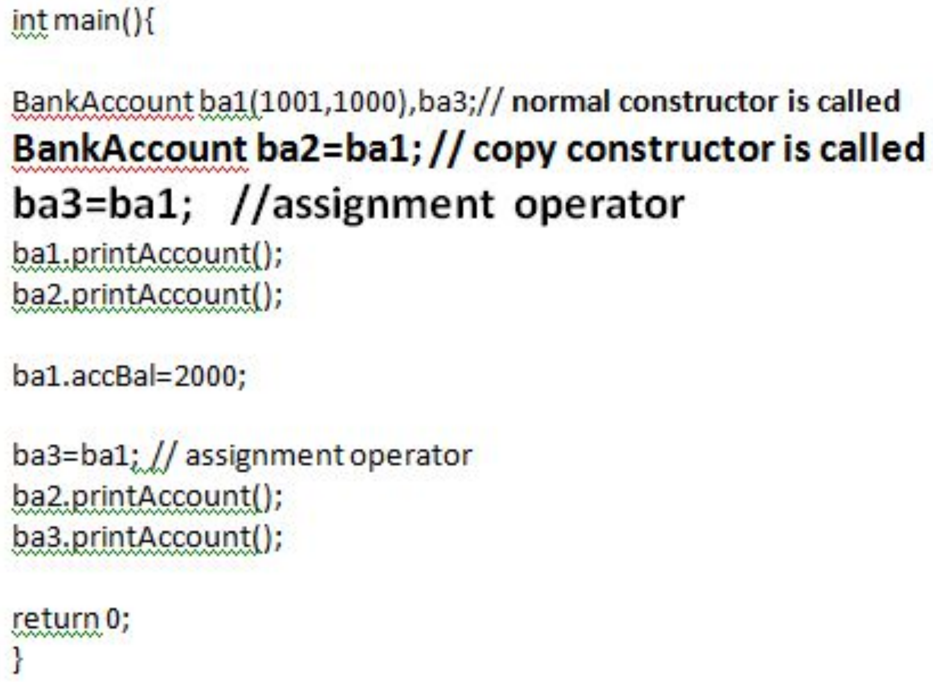
Copy Constructor

- **When is user-defined copy constructor needed?**
  - The C++ compiler creates a default copy constructor for each class which does a member-wise copy between objects.
  - The compiler created copy constructor works fine in general.
  - We need to define our own copy constructor only if an object has pointers or any runtime allocation of the resource like file handle, a network connection..etc.

Copy Constructor



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

class BankAccount{
public:
   int accNo;
   float accBal;
   BankAccount(){
   accNo=0;
   accBal=0;
   }
   BankAccount(int num, float bal){
   cout<<"parameterized constructo
   accNo=num;
   accBal=bal;
   }
   BankAccount(const BankAc
   cout<<"copy constructor is callec
   accNo=obj.accNo;
   accBal=obj.accBal;
   }
   void printAccount(){
   cout<<accNo<<":"<<accBal<<endl;
   }
};
```



```
int main(){

BankAccount ba1(1001,1000),ba3;// normal constructor is called
BankAccount ba2=ba1; // copy constructor is called
ba3=ba1;   //assignment  operator
ba1.printAccount();
ba2.printAccount();

ba1.accBal=2000;

ba3=ba1; // assignment operator
ba2.printAccount();
ba3.printAccount();

return 0;
}
```