Classes and Objects

Class is a way to bind the data describing an entity and its associated functions together.

```
class Account {  int accNo;
                 char type;
                 float balance;
                 public:
                 int a;
                 float deposit ( float amount)
                 { balance+=amount;
                   return balance;
                 }
                 float withdraw( float amount)
                 {balance - =amount;
                   return balance;
                 }
```

Here the data describing the class Account (accNo, type , balance) and its associated operations(deposit & withdraw) are bound together under one name Account

Need for classes

Classes are needed to represent real-world entities that not only have data type properties(their characteristics) but also have associated operations (their behavior).

```
class Student { int ID;
                 char grade;
                 float marks;
                 char retgrade()
                 { if (marks >75)
                   grade=‘A’;
                  if(marks>55)
                   grade=‘B’;
                 else
                  grade=‘C’;
                return grade;  }
             public:
              void accept(float m)
          { cin>> ID;  marks= m; }
            void display()
       { cout<<ID<<marks<<retgrade();}  };
```

- Student obj;
- cin>>m;
- obj.accept(m);

Declaration of classes

1. A class is a way to bind data and associated functions together.
2. The Internal data of a class is known as data members
3. The functions associated with the data is known as member functions.

Class members fall under one of the three different access permission category

- **Public member** – are accessible by all class users
- **Private member -** are only accessible by the class member within the class
- **Protected member –** are only accessible by the class members and members of the derived class.

Class Definition

- The general form of a class definition is as given below

```
class class-name
{  variable- declaration;
   function declaration;
   protected:
   variable- declaration;
   function declaration;
   public:
    variable- declaration;
     function declaration;
  };
```

By default the members in a class is private.

Q. Write the difference between class and structure. Explain with an example

Class Method definition

Member functions can be defined inside a class or outside a class:

```
class Account {  int accNo;
                   char type;
                   float balance;
                   public:
                   float deposit ( float amount)
                   { balance+=amount;
                     return balance;
                   }
                   float withdraw( float amount)
                   {balance - =amount;
                     return balance;
                   }
                };
```

// This is function defined inside the class

Member function defined outside class.

- Syntax

return type class-name:: function-name(parameter)

```
class Account {  int accNo;
                 char type;
                 float balance;
                 public:
                 float deposit ( float amount) ;
                 float withdraw( float amount);
 };

float  Account :: deposit ( float amount)
                 { balance+=amount;
                   return balance;
                 }
float Account :: withdraw( float amount)
                 {balance - =amount;
                   return balance;
                 }
```

Objects

- Objects are the variables of a user defined data type class.
- In other words, class acts as the data type
- and objects as its variable.

For eg :

To declare an object of class Amount is:

```
Amount  a;
```

Accessing class members

- The class members that are declared public can be accessed from outside the class
- ObjectName.FunctionName(Parameter);

Functions can be defined

- Inside the class
- Outside the class

```
#include<iostream.h>
class Transport
{  char Mode[20];
   char Name[20];
   void Get()
    { gets(Mode);  gets(Name); }
    void Show()
    {  cout<<Mode<<endl <<Name<<endl; }
}
void main()
{  transport T;
   T.Get();
    Show();
}
```

Q Define a class report with the following specification

Private member :

- admin, name, marks, average,
- getAvg()- to compute the average obtained in five subjects

Public member :

read() - function to accept values and invoke the function getAvg().

display() – function to display all the data members on the screen

Q Declare a class to represent bank account of 5 customer with the following data members.

Name of depositor, Account number, Type of account(S for saving & C for current), Balance amount.

The class should contain the member functions to do the following :

i. to Accept data members
ii. to deposit money
iii. to withdraw money (minimum balance should be 1000)
iv. to display the data members.

Q Define a class called Library for the following specification

Private member :

- Name as string
- eBook,pBook,mBook as int
- Price as float
- total as int
- retTotal calculates the number of all the books and returns the value should be stored in total

Public:

read() – accepts all the data.

calcTotal() – which invoke retTotal function and calculates the

Price	**total**
5000	**>45**
3500	**30-44**
2500	**15-29**
2000	**<15**

print() – prints all the data.

GLOBAL & LOCAL CLASS OBJECT

A class is said to be global class, if its definition occurs outside the bodies of all the functions in a program, which means that object of this class can be declared anywhere in the program.

```
class X          →  Global class
 {  ……
};


X obj;           →  Global object of class X
void main()
{  X obj1;}      →  Local  object obj1 only available in main
                    program
void fun()
{  X obj2; }     →  Local  object obj2 only available in function
                    fun
```

A class is said to be local class if its definition occurs inside a function body, which means that the object of this class type can be declared only within the function that defines this class type.

```
void main()
{  class Y           ——————→   Local class
    {   …..   };
  Y ob1;             ——————→  Local object ob1
}

void fun()
{  Y ob2; }          ——————→        Invalid
```

TYPES OF CLASS FUNCTIONS

ACCESSOR FUNCTION

These are the functions that allow us to access the data members. Accessor functions do not change the values of the data members.

MUTATOR FUNCTION

These are the member functions that allow us to change the data member of a class.

MANAGER FUNCTION

These are the member functions that deals with initializing and destroying class instances (ie constructors & destructors).

```
class Stud
{ int rno; char g; float mark;
  public:
   void read()
    { cin>>rno>>marks; }
  void display()
    {  cout<<rno<<mark<<g;}
 void calc()
 { if(marks>90)
   g=‘A’;
   :
   :
}
};
```

Accessor function :
read, display
Mutator function :
calc

Memory Allocation of Objects

- Member functions are created and placed in the memory space only once when the class is defined.
- Separate memory space is allocated to the objects at the time of their declaration for their data members only, because the data members hold different values for different objects.
- No separate space is allocated for member functions when the objects are created.

Nested class

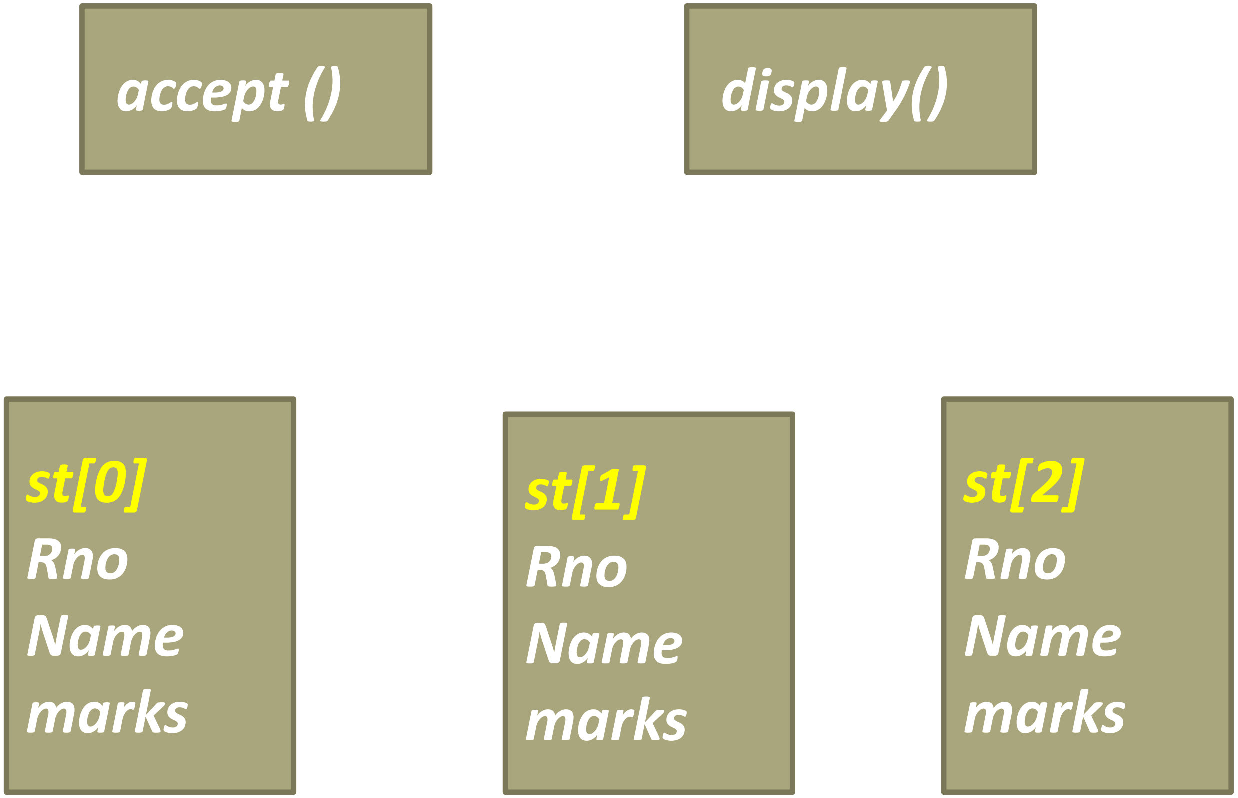
- A class may be declared within another class. A class declared within another is called a *nested class*.
- The outer class is known as the *enclosing class* and
- inner class is known as *nested class*.

```
#include<iostream.h>
class Outer { int a;
                    class Inner
                      { int b;
                        public:
                        int  c;
                        void prn()
                        {  cout<< " Inside Inner class prn() "<<endl;
                           cout<<b<<" \t"<<c<<endl;  }
                      };
```

ARRAY OF OBJECTS

```
class student
{ int rno ; char name[10]; float mark;
  public:
  void accept()
  { cin>> rno; gets(name); cin>> mark; }
  void display()
  { cout<<rno <<“ \t”<<name<<“\t”<<mark;
}
};
```

```
void main()
{ student st[10];
  int  n, i;
  cout<<“Enter the number of students :”;   cin>>n;
  for(i=0;i<n;i++)
   { cout<<“Enter the details of  :”<<i+1<<“student”;
        st[i].accept() ;           }
for(i=0;i<n;i++)
   { cout<<“Details of  :”<<i+1<<“student”;
      st[i].display() ;           }
```



accept ()

display()

st[0]
Rno
Name
marks

st[1]
Rno
Name
marks

st[2]
Rno
Name
marks

Highest mark

```
class student
{ int rno ; char name[10]; float mark;
  public:
  void accept()
  { cin>> rno; gets(name); cin>> mark; }
  void display()
  { cout<<rno <<“ \t”<<name<<“\t”<<mark; }
float retmark()
{ return mark }
};
```

```
void main()
{ student st[10];
  int  n, i;
  cout<<“Enter the number of students :”;   cin>>n;
  for(i=0;i<n;i++)
   { cout<<“Enter the details of  :”<<i+1<<“student”;
         st[i].accept();                                    }
for(i=0;i<n;i++)
   { cout<<“Details of  :”<<i+1<<“student”;
        st[i].display();                                    }
float lar=st[0].retmark();
int flag=0;
for(i=1;i<n;i++)
 { if(lar<st[i].retmark())
   { lar=st[i].retmark();
     flag=i; }
 }
cout<<“The student with maximum mark is”;
 st[flag].display();
}
```

search for a roll number

```
class student
{ int rno ; char name[10]; float mark;
  public:
  void accept()
  { cin>> rno; gets(name); cin>> mark; }
  void display()
  { cout<<rno <<“ \t”<<name<<“\t”<<mark;
}
float retmark()
{ return mark }
void search(int a)
{ if(a==rno)
  { cout<<“ Roll number found  :”
   display();
   return(1);
   }
 return 0;
}
};
```

```
void main()
{ student st[10];
  int  n, i;
  cout<<“Enter the number of students :”;   cin>>n;
  for(i=0;i<n;i++)
   { cout<<“Enter the details of  :”<<i+1<<“student”;
        st[i].accept();                                  }
for(i=0;i<n;i++)
   { cout<<“Details of  :”<<i+1<<“student”;
          st[i].display();                               }
int num, flag=0;
cout<<“Enter the roll number to be searched “;
cin>>num;
for(i=0;i<n;i++)
{ if( st[i].search(num)==1)
     { flag=1;  break; }  }
if(flag==0)
  cout<<“Roll number doesnot exist “;
}
```

- Program search for a roll number and change the mark of that roll number.

- Program to enter unique roll number (no repetition of roll number)

TYPES OF CLASS FUNCTIONS

ACCESSOR FUNCTION

These are the functions that allow us to access the data members. Accessor functions do not change the values of the data members.

MUTATOR FUNCTION

These are the member functions that allow us to change the data member of a class.

MANAGER FUNCTION

These are the member functions that deals with initializing and destroying class instances (ie constructors & destructors).

Nested Class

- A class may be defined within another class. this is known as nested class.
- The outer class is known as enclosing class and the inner class is known as nested class.

Nested class

```
#include<iostream.h>
class Outer { int a;
              class Inner
                 { int b;
                  public:
                  int  c;
                 void assign()
                 { b=5;  c=10; }
                  void prn()
                  {  cout<< “ Inside Inner class prn() “<<endl;
                    cout<<b<<“ \t”<<c<<endl;  }
               };
        Inner ob1;
        public:
        Inner ob2;
       void assign2()
       { a=25;}
        void second()
        { cout<<“ In the Outer class “<<endl;
          cout<< ob2.b<<ob2.c<<a<<endl; }
 };
```

```
void main()
 {  Outer ab;
   ab.assign2();
   ab.ob2.assign();
   ab.second()
   ab.ob2.prn()
}
```

output:

- All classes including enclosing classes and nested classes obey the usual access rule.
- The member functions of a nested class have no special access to members of an enclosing class.
- That is the nested class can access public member of its enclosing class using an object only.
- The access to private and protected member is not available to other class

Data Hiding and Encapsulation

- The part of the class that is been hidden from the outside world(i.e private and protected) supports **data hiding.**

- **Abstraction** refers to the act of showing only the essential features without including the background details or explanation(i.e the public area in a class)

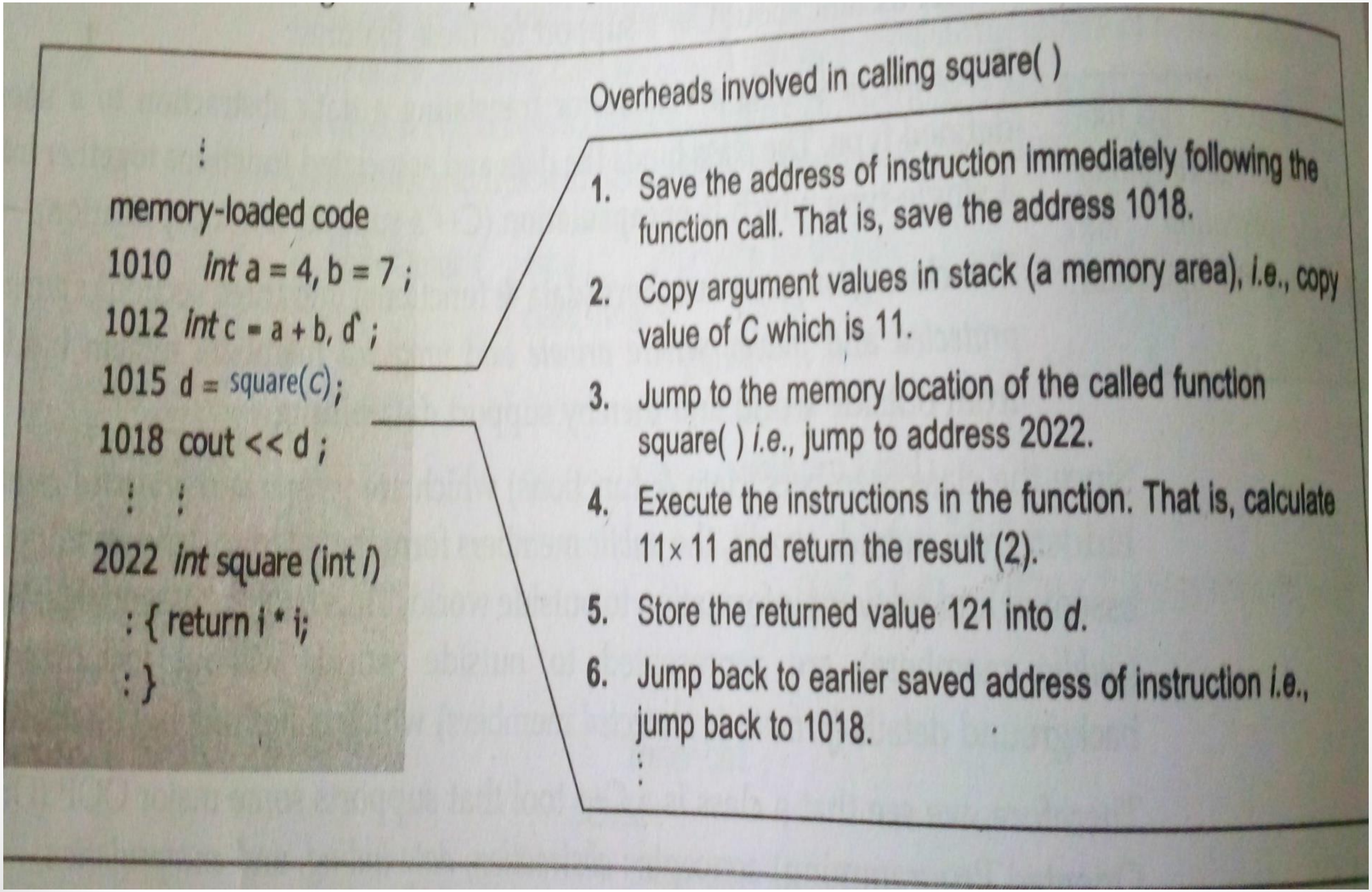
- **Abstraction supports data hiding so that only the relevant information is exposed to the user and the rest of the information is hidden from the user.**

- **Encapsulation** is the wrapping up of data and the associated member functions into one unit.

Inline functions

- The inline functions are designed to speed up the programs
- The coding of normal function and inline functions are similar except that inline function definition start with the keyword inline.

Working of a normal function



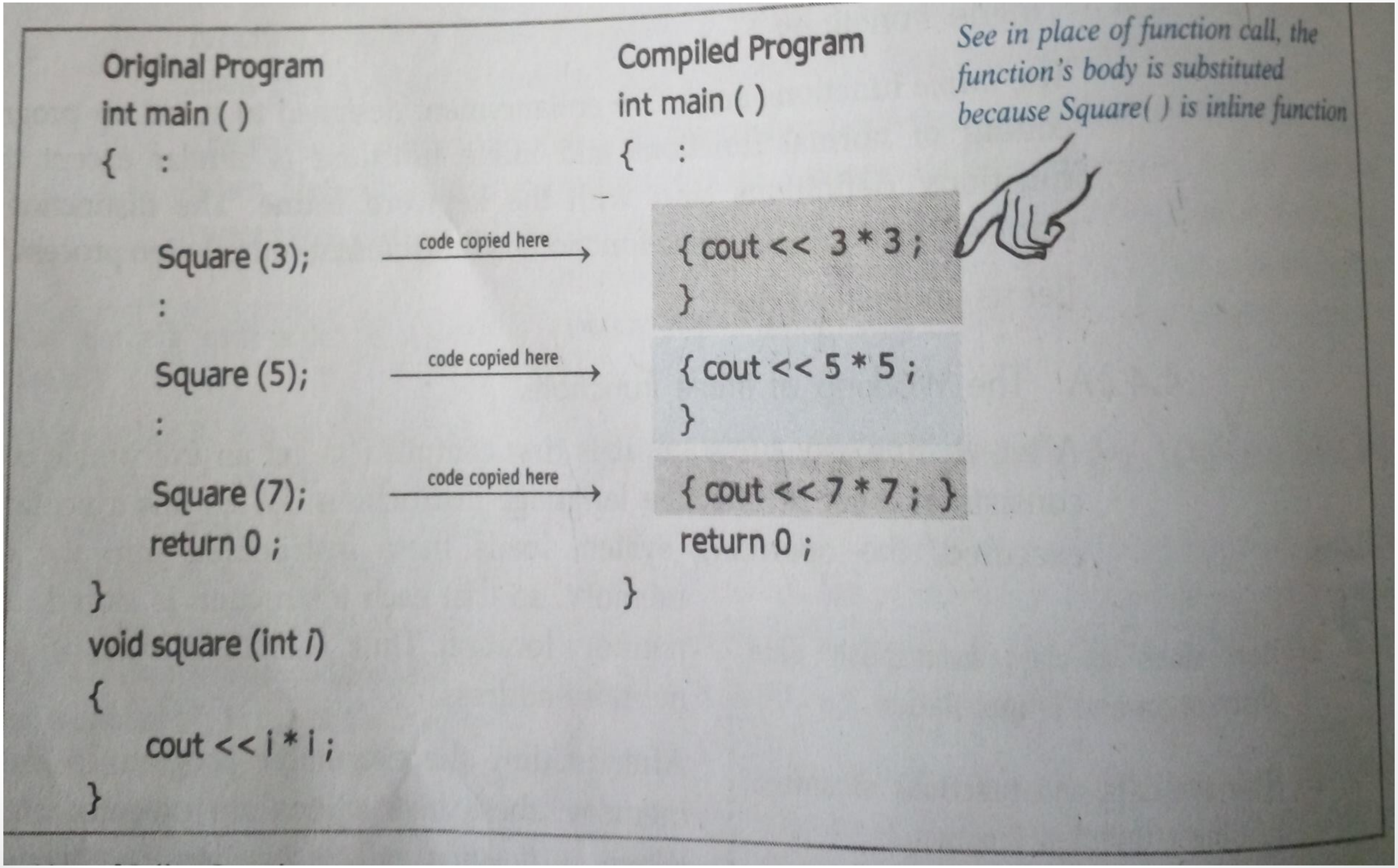
```
  ⋮
memory-loaded code
1010  int a = 4, b = 7 ;
1012  int c = a + b, d ;
1015  d = square(c);
1018  cout << d ;
 ⋮   ⋮
2022  int square (int i)
  : { return i * i;
  : }
```

Overheads involved in calling square( )

1. Save the address of instruction immediately following the function call. That is, save the address 1018.
2. Copy argument values in stack (a memory area), *i.e.*, copy value of C which is 11.
3. Jump to the memory location of the called function square( ) *i.e.*, jump to address 2022.
4. Execute the instructions in the function. That is, calculate 11× 11 and return the result (2).
5. Store the returned value 121 into *d*.
6. Jump back to earlier saved address of instruction *i.e.*, jump back to 1018.

⋮

Working of inline function



Original Program

```
int main ( )
{   :

     Square (3);
     :

     Square (5);
     :

     Square (7);
     return 0 ;
}
void square (int i)
{
     cout << i * i ;
}
```

code copied here

code copied here

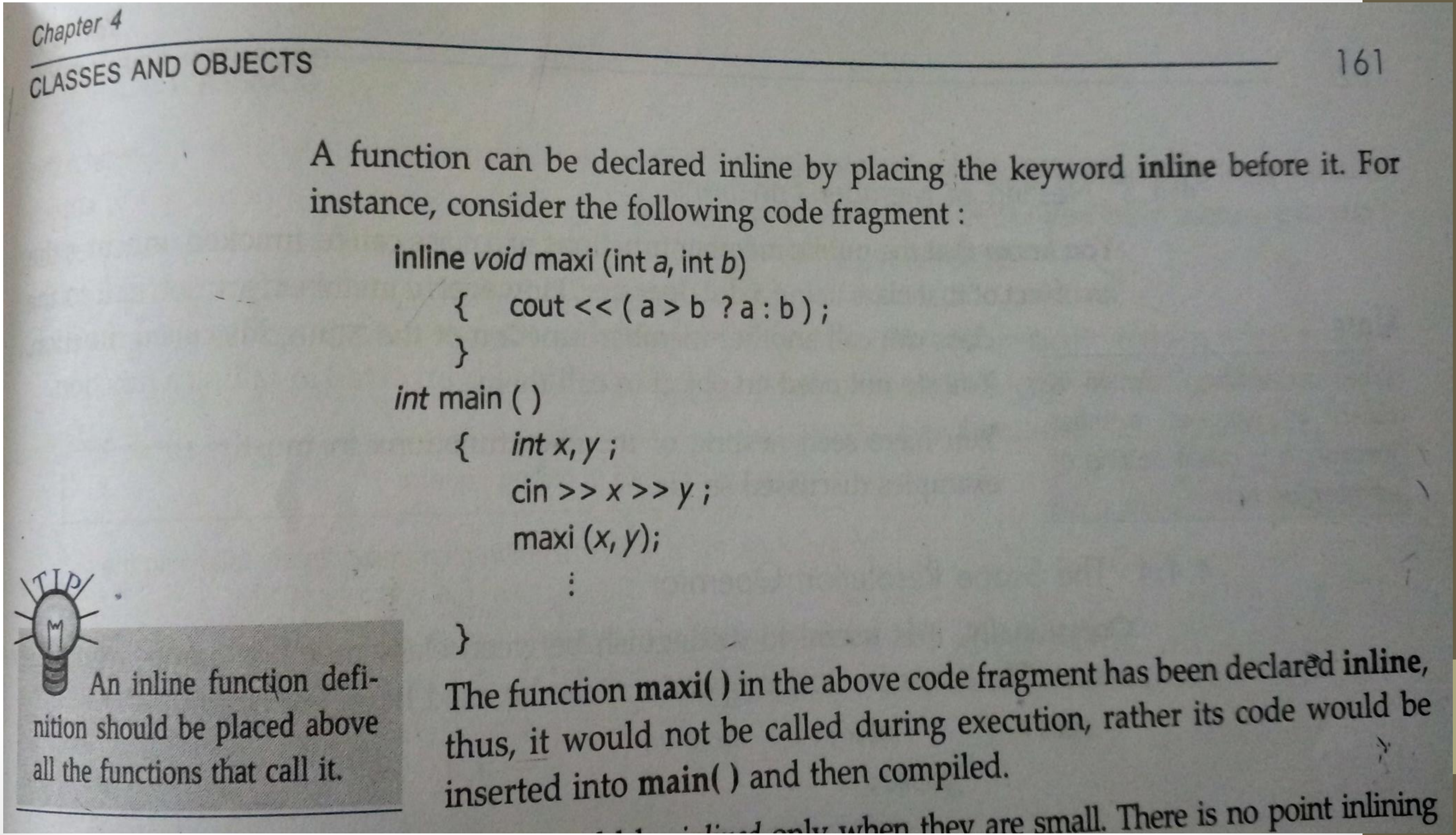
code copied here

Compiled Program

```
int main ( )
{   :

    { cout <<  3 * 3 ;
    }

    { cout << 5 * 5 ;
    }

    { cout << 7 * 7 ;  }
    return 0 ;
}
```

See in place of function call, the function's body is substituted because Square( ) is inline function

How to define an inline function?



A function can be declared inline by placing the keyword **inline** before it. For instance, consider the following code fragment :

```
inline void maxi (int a, int b)
    {   cout << ( a > b ? a : b ) ;
    }
int main ( )
    {   int x, y ;
        cin >> x >> y ;
        maxi (x, y);
           :
    }
```

TIP

An inline function definition should be placed above all the functions that call it.

The function **maxi( )** in the above code fragment has been declared **inline**, thus, it would not be called during execution, rather its code would be inserted into **main( )** and then compiled.

only when they are small. There is no point inlining

- The function should be inlined only if the code is small
- The function inlining will not work in the following cases

1. For functions that return values and have a loop or a switch or a goto.
2. If the functions contain static variables.
3. If the function is a recursive (a function that calls itself)

Constant member function

- If the member function of a class does not alter any data in the class, then this member function is called as constant member function using the keyword const.

eg

```
int maxi(int, int) const;
void prn() const;
```

The qualifier const will appear in both the function declaration and definition.

Nesting of member function

- When the member function of the same class calls another member function of the same class then it is called nesting of member functions.

Objects as Function Arguments

- There are two ways to pass the objects
- Call by value
- call by reference

Call by value

```
class Time { int h,m,s;
                public:
                 void get_time(int hr,int min,int sec)
                 { h=hr;  m=min; s=sec; }

                   void put_time()
                   { cout<<h<< “:” <<m<< “:” <<s; }
                   int gethr(){ return h;}
                   int getmin() { return m;}
                   int getsec() { return s;}
};
```

```
void sum(Time t1, Time t2);
            // add both the time and store in T1

void convert(Time t1,char ch);
          //time in hr or time in am/pm
```

```
void main()
{  Time tm1, tm2;
   tm1.get_time(5,12,45);
   tm2. get_time(7,2,15);
    convert(tm1,ch);
    sum(tm1,tm2);
      prnvalues(tm1);
}
```

Call by Reference

```
class Time { int h,m,s;
             public:
              void get_time(int hr,int min,int sec)
              { h=hr;  m=min; s=sec; }

                void put_time()
                { cout<<h<<  “:”  <<m<<  “:”  <<s; }
                int gethr()    { return h;}
                int getmin()  { return m;}
                int getsec()  { return s;}
};
```

```
void sum(Time &t1, Time &t2);
        // add both the time and store in T1

void convert(Time &t1,char ch);
       //time in hr or time in am/pm
```

```
void main()
{  Time tm1, tm2;
   tm1.get_time(5,12,45);
   tm2. get_time(7,2,15);
    convert(tm1,ch);
    sum(tm1,tm2);
      prnvalues(tm1);
}
```

Function returning an object

```
void convert(Time &t1,char ch);
        //time in hr or time in am/pm

Time convert ( Time &t1, char ch)
  { ..................
     return (t1) ;   }
```

Function call :

```
 Time T=convert(t1,ch);
```