Appendix A

Projects

A.1 Minor Project 1: Menu Based Calculation System

Learning Objectives

The designing of the Menu Based Calculation System project will help the students to:

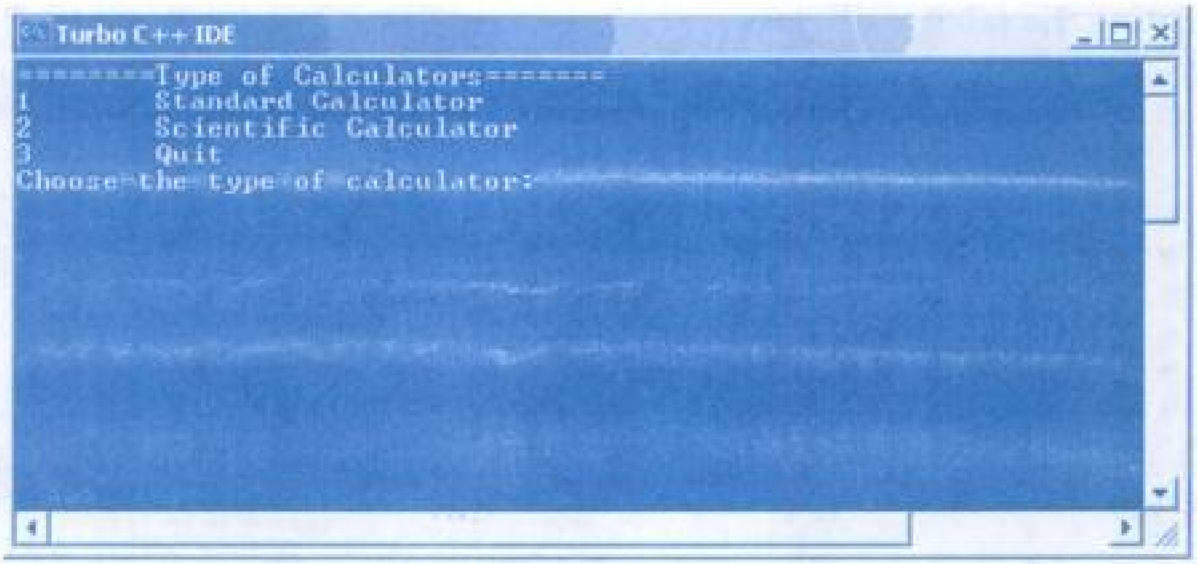
- Create C++ classes with static functions
- Generate and call static functions
- Use the functions of **Math.h** header file
- Develop and display the main menu and its submenus

Understanding the Menu Based Calculation System

The Menu Based Calculation System project is aimed at performing different types of calculations including normal and scientific calculations. In this project, two calculators, Standard and Scientific, are used for performing the calculations. The Standard calculator helps in performing simple calculations such as addition, multiplication, etc. while the Scientific calculator helps in performing mathematical operations such as finding the square or cube of a number.

The first screen contains a menu from which you can select the type of calculator: Standard, or Scientific. The first screen also provides the Quit option to terminate the execution of the application. Figure A.1 shows the first screen of the menu based calculation system.

To select a calculator, enter the integer corresponding to the calculator name. For instance, if you select **1**, the Standard calculator will open up, while selecting **2** will open the Scientific calculator.



```
========Type of Calculators=======
1       Standard Calculator
2       Scientific Calculator
3       Quit
Choose the type of calculator:
```

Fig. A.1

Developing the Menu Based Calculation System

The code of the calculator application mainly comprises of two classes **stand_calc** and **scien_calc**. The stand_calc class helps to perform standard calculations. The scien_calc class, on the other hand, helps to perform scientific calculations. Both classes contain static functions so as to ensure that these functions can be called in the main function through class name.

Creating the stand_calc class

The stand_calc class aims at performing specific tasks related to standard calculations. These tasks are:

- Adding two numbers
- Subtracting the second number from the first number
- Multiplying two numbers
- Dividing the first number by the second number
- Modulus of the first number by the second number

To perform the above-mentioned tasks, the stand_calc class implements the following member functions:

Functions	*Description*
Addition	Returns the addition of two input numbers.
Subtraction	Returns the subtraction of two numbers accepted as input from the user.
Multiplication	Returns the multiplication of two numbers accepted as input from the user.
Division	Returns the output obtained after performing the division operation on the input numbers.
Modulus	Returns the output obtained after performing the modulus operation on the input numbers.

Creating the scien_calc class

You need to create the scien_calc class to perform tasks related to scientific calculations, which include finding the square or cube of a number, etc. The scien_calc class performs the following tasks:

- Determines the square of a number
- Determines the cube of a number
- Determines the first number to the power of the second number
- Determines the square root of a number
- Determines the factorial of a number
- Determines the value of sin, cos and tan by passing a number

To perform the above-mentioned tasks, the scien_calc class implements the following member functions:

Functions	*Description*
Square	Accepts a number and returns the square of that number
Cube	Accepts a number and returns the cube of that number
Power	Accepts two numbers and returns the first number to the power of the second number
sq_root	Accepts a number and returns its square root
Fact	Returns the factorial of an input number
sin_func	Returns the sin value of an input number
cos_func	Returns the cos value of an input number
tan_func	Returns the tan value of an input number

Calc

```
/* calc.cpp is a calculator. Initially, it displays a main menu to choose the calculator
type. If a user chooses Standard calculator, then a menu appears for standard calculator
options. If a user chooses Scientific calculator, then a menu appears for scientific
calculator options and the last option is to Quit.
In standard calculator, options are to add, subtract, multiply etc. and in scientific
calculator, options are power, factorial, square root, etc.
In this program, preprocessor are defined for new calculation and old calculation. New
calculation will accept an operand whereas in old calculation, one operand is already
assumed from the result of previous calculation.
Exception handling is not implemented in this project, so do not enter a string when
system asks you for a number.
    */
    //File including and preprocessor declaration
    #include <iostream.h>
    #include <conio.h>
    #include <math.h>
```

```
#include <stdlib.h>
#define new_cal 1
#define old_cal 0
//stand_calc class to define standard calculator functions
class stand_calc
{
  /*Protyping of standard calculator functions. These functions are static, therefore
calling of these functions is possible with the name of the class. There is no need
to create an object of the class. */
  public:
  static double addition(double,double);
  static double subtract(double,double);
  static double multiplication(double,double);
  static double division(double ,double *);
  static double modulus(double *,double *);
};
//scien_calc class to define scientific calculator functions
class scien_calc
{
  public:
  static double square(double);
  static double cube(double);
  static double power(double,double);
  static double sq_root(double);
  static long int fact(double);
  static double sin_func(double);
  static double cos_func(double);
  static double tan_func(double);
};
//addition function will add two numbers
double stand_calc::addition(double a, double b)
{
  return(a+b);
}
//subtract function will subtract the second number from the first number
double stand_calc::subtract(double a, double b)
{
  return(a-b);
}
//multiplication function will multiply two numbers
double stand_calc::multiplication(double a, double b)
{
  return(a*b);
}
```

```
/*division function will divide the first number by the second number. This function
accepts two arguments, one is copy of a variable and another is pointer type because
if accepting divisor is zero, then this function will show a message to enter the
divisor again. Using pointer means that the entered value of the divisor for this
function should be updated at the main function also.*/
double stand_calc::division(double a, double *b)
{
   while(*b==0)
   {
         cout<<"\nCannot divide by zero.";
         cout<<"\nEnter second number again:";
         cin>>*b;
   }
   return(a/(*b));
}
/*Modulus function will divide the first number by the second number and return the
remainder part of the division. Similar to division function, it will not accept
zero in the divisor. Modulus cannot be performed on a double number, so we need to
convert it into an integer.*/
double stand_calc::modulus(double *a, double *b)
{
   while(*b==0)
   {
         cout<<"\nCannot divide by zero.";
         cout<<"\nEnter second number again:";
         cin>>*b;
   }
   //Converting double into an integer
   int x=(int)*a;
   int y=(int)*b;
   if(*a-x>0||*b-y>0)
         cout<<"\nConverting decimal number into an integer to perform modulus";
   *a=x;
   *b=y;
   return(x%y);
}
//Declaration of scien_calc class functions starts from here.
//square function of scien_calc class to return accepting number to the power 2
double scien_calc::square(double x)
{
   return(pow(x,2));
}
//cube function of scien_calc class to return accepting number to the power 3
double scien_calc::cube(double x)
{
   return(pow(x,3));
}
```

```
//power function of scien_calc class to return the first number to the power of the
second number
double scien_calc::power(double x,double y)
{
   return(pow(x,y));
}
//sq_rrot function of scien_calc class to return the square root of the entered number
double scien_calc::sq_root(double x)
{
   return(sqrt(x));
}
/*fact function of the scien_calc class to return a long integer as factorial of an
accepting number. This will convert accepting number into an integer before calculating
the factorial*/
long int scien_calc::fact(double x)
{
   int n=(int)x;
   long int f=1;
   while(n>1)
   {
         f*=n;
         n--;
   }
   return f;
}
//sin_func of the scien_calc class to return the sin value of x
double scien_calc::sin_func(double x)
{
   return(sin(x));
}
//cos_func of the scien_calc class to return the cos value of x
double scien_calc::cos_func(double x)
{
   return(cos(x));
}
//tan_func of the scien_calc class to return the tan value of x
double scien_calc::tan_func(double x)
{
   return(tan(x));
}

//Displaying the menus to enter the options and values
void main()
{
   double num1,num2,num3,temp;
   int choice1=0,choice2,flag;
   //Loop of main menu from where the program starts. It will show the menu to choose
the type of calculator.
```

```
    do
    {
            clrscr();
            cout<<"========Type of Calculators=======";
            cout<<"\n1\tStandard Calculator\n2\tScientific Calculator\n3\tQuit";
            cout<<"\nChoose the type of calculator:";
            cin>>choice1;
            flag=new_cal;
            //To perform an operation according to the entered option in the main menu
            switch(choice1)
            {
                    case 1:
                            //Loop to display the standard calculator menu
                            do
                            {
                                    clrscr();
                                    cout<<"==========Standard Calculator===========";

cout<<"\n1\tAddition\n2\tSubtraction\n3\tMultiplication\n4\tDivision\n5\tModulus\n6\tReturn
to Previous Menu\n7\tQuit";
                                    //Option 8 will be displayed only when working on
old calculations. Here, already a number is saved in the calculator memory.
                                    if(flag==old_cal)
                                            cout<<"\n8\tClear Memory";
                                    cout<<"\nChoose the type of calculation:";
                                    cin>>choice2;
                                    //To perform operation and call functions of the
stand_calc class
                                    switch(choice2)
                                    {
                                            case 1:
                                                    //If a new calculation is there, then
accept the first number else previous calculation result will be the first number.
                                                    if (flag==new_cal)
                                                    {
                                                            cout<<"Enter first number:";
                                                            cin>>num1;
                                                    }
                                                    else
                                                    {
                                                            num1=temp;
                                                            cout<<"\nFirst number is
"<<num1<<endl;
                                                    }
                                                    cout<<"Enter second number:";
                                                    cin>>num2;
```

```
num3=stand_calc::addition(num1,num2);
                                                cout<<"\nAddition of "<<num1<<" and
"<<num2<<" is "<<num3;
                                                cout<<"\nPress  any  key  to
continue..........";
                                                getch();
                                                temp=num3;
                                                flag=old_cal;
                                                break;
                                        case 2:
                                                if (flag==new_cal)
                                                {
                                                        cout<<"Enter first number:";
                                                        cin>>num1;
                                                }
                                                else
                                                {
                                                        num1=temp;
                                                        cout<<"\nFirst  number  is
"<<num1<<endl;
                                                }
                                                cout<<"Enter second number:";
                                                cin>>num2;

num3=stand_calc::subtract(num1,num2);
                                                cout<<"\nSubtraction of "<<num2<<"
from "<<num1<<" is "<<num3;
                                                cout<<"\nPress  any  key  to
continue..........";
                                                getch();
                                                temp=num3;
                                                flag=old_cal;
                                                break;
                                        case 3:
                                                if (flag==new_cal)
                                                {
                                                        cout<<"Enter first number:";
                                                        cin>>num1;
                                                }
                                                else
                                                {
                                                        num1=temp;
                                                        cout<<"\nFirst  number  is
"<<num1<<endl;
                                                }
                                                cout<<"Enter second number:";
                                                cin>>num2;
```

```
num3=stand_calc::multiplication(num1,num2);
                                                            cout<<"\nMultiplication of "<<num1<<"
and "<<num2<<" is "<<num3;
                                                            cout<<"\nPress  any  key  to
continue..........";
                                                            getch();
                                                            temp=num3;
                                                            flag=old_cal;
                                                            break;
                                                    case 4:
                                                            if (flag==new_cal)
                                                            {
                                                                    cout<<"Enter first number:";
                                                                    cin>>num1;
                                                            }
                                                            else
                                                            {
                                                                    num1=temp;
                                                                    cout<<"\nFirst  number  is
"<<num1<<endl;
                                                            }
                                                            cout<<"Enter second number:";
                                                            cin>>num2;

num3=stand_calc::division(num1,&num2);
                                                            cout<<"\nDivision of "<<num1<<" by
"<<num2<<" is "<<num3;
                                                            cout<<"\nPress  any  key  to
continue..........";
                                                            getch();
                                                            temp=num3;
                                                            flag=old_cal;
                                                            break;
                                                    case 5:
                                                            if (flag==new_cal)
                                                            {
                                                                    cout<<"Enter first number:";
                                                                    cin>>num1;
                                                            }
                                                            else
                                                            {
                                                                    num1=temp;
                                                                    cout<<"\nFirst  number  is
"<<num1<<endl;
                                                            }
                                                            cout<<"Enter second number:";
                                                            cin>>num2;
```

```
num3=stand_calc::modulus(&num1,&num2);
                                                cout<<"\nModulus of "<<num1<<" by
"<<num2<<" is "<<num3;
                                                cout<<"\nPress any key to
continue..........";
                                                getch();
                                                temp=num3;
                                                flag=old_cal;
                                                break;
                                        case 6:
                                                cout<<"\nReturning to previous menu.";
                                                cout<<"\nPress any key to
continue..........";
                                                getch();
                                                break;
                                        case 7:
                                                cout<<"\nQuitting............";
                                                cout<<"\nPress any key to
continue...........";
                                                getch();
                                                exit(0);
                                        case 8:
                                                //If a new calculation is going on
then 8 is an invalid option, else 8 is an option to start a new calculation
                                                if(flag==new_cal)
                                                {
                                                        cout<<"\nInvalid choice.";
                                                        cout<<"\nPress any key to
continue.........";
                                                        getch();
                                                }
                                                else
                                                {
                                                        temp=0;
                                                        flag=new_cal;
                                                }
                                                break;
                                        default:
                                                cout<<"\nInvalid choice.";
                                                cout<<"\nPress any key to
continue.............";
                                                getch();
                                                break;
                                }
                        }while (choice2!=6);
                        break;
```

```
                case 2:
                        //Loop to display scientific calculator menu
                        do
                        {
                                clrscr();
                                cout<<"==========Scientific Calculator===========";
cout<<"\n1\tSquare\n2\tCube\n3\tPower\n4\tFactorial\n5\tSin\n6\tCos\n7\tTan\n8\tReturn
to previous menu\n9\tQuit";
                                if(flag==old_cal)
                                        cout<<"\n10\tClear Memory";
                                cout<<"\nChoose the type of calculation:";
                                cin>>choice2;
                                switch(choice2)
                                {
                                        case 1:
                                                if (flag==new_cal)
                                                {
                                                        cout<<"Enter number to find
square:";
                                                        cin>>num1;
                                                }
                                                else
                                                {
                                                        num1=temp;
                                                        cout<<"\nNumber     is
"<<num1<<endl;
                                                }
                                                num3=scien_calc::square(num1);
                                                cout<<"\nSquare of "<<num1<<" is
"<<num3;
                                                cout<<"\nPress  any  key  to
continue..........";
                                                getch();
                                                temp=num3;
                                                flag=old_cal;
                                                break;
                                        case 2:
                                                if (flag==new_cal)
                                                {
                                                        cout<<"Enter number to find
cube:";
                                                        cin>>num1;
                                                }
                                                else
                                                {
                                                        num1=temp;
                                                        cout<<"\nNumber     is
```

```
"<<num1<<endl;
                                                }
                                                num3=scien_calc::cube(num1);
                                                cout<<"\nCube of "<<num1<<" is
"<<num3;
                                                cout<<"\nPress any key to
continue..........";
                                                getch();
                                                temp=num3;
                                                flag=old_cal;
                                                break;
                                        case 3:
                                                if (flag==new_cal)
                                                {
                                                       cout<<"Enter first number
for base to find power:";
                                                       cin>>num1;
                                                }
                                                else
                                                {
                                                       num1=temp;
                                                       cout<<"\nFirst number is
"<<num1<<endl;
                                                }
                                                cout<<"Enter second number for power
to find power:";
                                                cin>>num2;
                                                num3=scien_calc::power(num1,num2);
                                                cout<<"\n"<<num1<<" to the power
"<<num2<<" is "<<num3;
                                                cout<<"\nPress any key to
continue..........";
                                                getch();
                                                temp=num3;
                                                flag=old_cal;
                                                break;
                                        case 4:
                                                if (flag==new_cal)
                                                {
                                                       cout<<"Enter number to find
factorial:";
                                                       cin>>num1;
                                                }
                                                else
                                                {
                                                       num1=temp;
                                                       cout<<"\nNumber to find
```

```
factorial is "<<num1<<endl;
                                      }
                                      long int num4=scien_calc::fact(num1);
                                      cout<<"\nFactorial of "<<num1<<" is
"<<num4;
                                      cout<<"\nPress any key to
continue..........";
                                      getch();
                                      temp=num4;
                                      flag=old_cal;
                                      break;
                                case 5:
                                      if (flag==new_cal)
                                      {
                                            cout<<"Enter number to find
sin value:";
                                            cin>>num1;
                                      }
                                      else
                                      {
                                            num1=temp;
                                            cout<<"\nNumber for sin value
is "<<num1<<endl;
                                      }
                                      num3=scien_calc::sin_func(num1);
                                      cout<<"\nSin value of "<<num1<<" is
"<<num3;
                                      cout<<"\nPress any key to
continue..........";
                                      getch();
                                      temp=num3;
                                      flag=old_cal;
                                      break;
                                case 6:
                                      if (flag==new_cal)
                                      {
                                            cout<<"Enter number to find
cos value:";
                                            cin>>num1;
                                      }
                                      else
                                      {
                                            num1=temp;
                                            cout<<"\nNumber for cos value
is "<<num1<<endl;
                                      }
                                      num3=scien_calc::cos_func(num1);
                                      cout<<"\nCos value of "<<num1<<" is
```

```
"<<num3;
                                                cout<<"\nPress any key to

continue..........";

                                                getch();
                                                temp=num3;
                                                flag=old_cal;
                                                break;
                                        case 7:
                                                if (flag==new_cal)
                                                {
                                                        cout<<"Enter number to find

tan value:";

                                                        cin>>num1;
                                                }
                                                else
                                                {
                                                        num1=temp;
                                                        cout<<"\nNumber for tan value

is "<<num1<<endl;

                                                }
                                                num3=scien_calc::tan_func(num1);
                                                cout<<"\nTan value of "<<num1<<" is

"<<num3;

                                                cout<<"\nPress any key to

continue..........";

                                                getch();
                                                temp=num3;
                                                flag=old_cal;
                                                break;
                                        case 8:
                                                cout<<"\nReturning to previous menu.";
                                                cout<<"\nPress any key to

continue..........";

                                                getch();
                                                break;
                                        case 9:
                                                cout<<"\nQuitting.............";
                                                cout<<"\nPress any key to
continue...........";

                                                getch();
                                                exit(0);
                                        case 10:
                                                if(flag==new_cal)
                                                {
                                                        cout<<"\nInvalid choice.";
                                                        cout<<"\nPress any key to
continue........";
                                                        getch();
                                                  }
                                                  else
                                                  {
                                                        temp=0;
                                                        flag=new_cal;
                                                  }
                                                  break;
                                            default:
                                                  cout<<"\nInvalid choice.";
                                                  cout<<"\nPress any key to
continue............";
                                                  getch();
                                                  break;
                                      }
                                }while (choice2!=8);
                                break;
                          case 3:
                                cout<<"\nQuitting......";
                                cout<<"\nPress any key to continue.......";
                                getch();
                                break;
                          default:
                                cout<<"\nInvalid Choice.";
                                cout<<"\nPress any key to continue.......";
                                getch();
                                break;
                    }
      }while (choice1!=3);
    }
```

A.2 Major Project 1: Banking System

Learning Objectives

The designing of the Banking System project helps the students to:

- Create C++ classes and call the functions declared in the classes
- Develop and display main menu and its submenus
- Change the menu options during runtime
- Programmatically create files using File System objects
- Perform file transactions such as Updation, Deletion and Display from files
- Use **iomanip** header file in C++ to display formatted output of data using **setw()** function for setting width of the text to be displayed.

Hidden page

Creating the dispRecords Class

You need to create the **dispRecords** class to implement the functionality of displaying the information related to the customers of a bank and their accounts. In the dispRecords class, data related to customers is retrieved from the newrecords.dat data file for displaying customer information or adding and closing of customer accounts. You can create the dispRecords class by defining the variables required for displaying customer and account information and the member functions such as **displayCustomer** and **deleteAccount**. The following table lists the member functions that need to be defined in the class dispRecords:

Functions	*Descriptions*
addDetails(int, char name[30], char address[60], float)	Adds the information related to a new customer of the bank who becomes an account holder.
displayCustomers(void)	Displays a list of all the account holders of the bank along with their account numbers and balance.
deleteAccount(int)	Deletes the information related to the account holder from the newrecords.dat data file.
updateBalance(int, float)	Updates the balance after a customer has performed a deposit or withdrawal transaction.
lastAccount()	Displays the account number of the last entry.
accountExists(int)	Checks whether an account exists or not.
getName(int)	Retrieves the name of the account holder.
getAddress(int)	Retrieves the address of the account holder.
getBalance(int)	Retrieves the balance of the account holder.
getRecord(int)	Returns the record number from the newrecords.dat data file when an employee of the bank enters the account number related to an account holder.
display(int)	Displays all the information related to an account holder from the newrecords.dat file on the basis of specified account number.

Creating the accountTransactions Class

You need to create the **accountTransactions** class so that transactions related to an account can be performed. The data related to the transactions are stored in the transaction.dat data file. The accountTransactions class also uses some member functions defined in the dispRecords class. In the class accountTransactions, the Object Oriented Programming (OOP) concepts of Polymorphism are used to manipulate data, which need to be stored in the transaction.dat data file. You can create the accountTransactions class by defining variables and member functions, which include **new_account** and **showAccount**. The following table lists the member functions of the accountTransactions class:

Functions	*Descriptions*
new_account(void)	Validates the information related to a new customer and adds the information to the transaction.dat data file using the addDetails member function.
closeAccount()	Closes the account of an account holder after verifying the account number.
showAccount(int)	Displays the headings Customer Name, Deposit and Withdrawal, Interest and Balance.
display_account(void)	Displays the data related to a specific account holder.
deleteAccount(int)	Deletes the data related to a transaction from the transaction.dat data file on the basis of the account number of that account holder.
transaction(void)	Helps to perform deposit and withdrawal transactions.
dateDiffer(int, int, int, int, int, int)	Checks the current and account creation dates. If the account in the bank has completed one year, then interest for that account is calculated.
getInterest(int, float)	Generates interest when one year has completed for a particular account.
showInterest(void)	Displays the interest generated using the getInterest member function. The showInterest member function also helps to update the balance of the account holder.

Banking_Application

```
/** A Banking System with normal transactions **/
    #include <iostream.h>
    #include <fstream.h>
    #include <process.h>
    #include <string.h>
    #include <stdlib.h>
    #include <stdio.h>
    #include <ctype.h>
    #include <conio.h>
    #include <dos.h>
    #include <iomanip.h>

    // The Menus Class displays the Menu
    class Menus
    {
      public :
                    void showmenu(void) ;
```

```
   private :
                  void closemenu(void) ;
};

// The Class displays all the Customer Account related functions
class dispRecords
{
   public :
                  void addDetails(int, char name[30], char address[60], float) ;
                  void displayCustomers(void) ;
                  void deleteAccount(int) ;
                  void updateBalance(int, float) ;
                  void updateCustomer(void) ;
                  int  lastAccount(void) ;
                  int  accountExists(int) ;
                  char *getName(int);
                  char *getAddress(int);
                  float getBalance(int) ;
                  int  getRecord(int) ;
                  void display(int) ;
                  void  displayList(void) ;
                  int   AccountNumber ;
                  char  name[50], address[50] ;
                  float intBalance ;
};

// The Class has all the transaction related methods
class accountTransactions
{
   public :
                  void new_account(void);
                  void closeAccount(void);
                  void display_account(void);
                  void transaction(void);
                  void addDetails(int, int, int, int, char, char typeTransaction[15],
float, float, float);
                  void  deleteAccount(int);
                  int   dateDiffer(int, int, int, int, int, int);
                  float getInterest(int, float);
                  void  display(int);
                  void  showAccount(int);
                  int   AccountNumber;  //variable for Account Number
                  char  trantype[10];  // variable of cheque or cash input or output
                  int   dday, mmonth, yyear;  // transaction date
                  char  transactions;        // type of transactions - Deposit or
Withdrawal of Amount
```

```
                float intInterest, intAmount, intBalance;
                static float calcInterest;
                void showInterest(void);//added
};

// showmenu() method to display the Main Menu in the application
void Menus :: showmenu(void)
{
   char choice;
   while (1)
   {
          clrscr();
          cout<<"\n      --Welcome to Banking System Application-     \n";
          cout<<" ******************************************\n\n";
          cout<<"        Choose from Options \n";
          cout<<" ________________ \n";
          cout <<"       1: Open an Account\n";
          cout <<"       2: View an Account      \n";
          cout <<"       3: Show all Accounts    \n";
          cout <<"       4: Make a Transaction  \n";
          cout <<"       5: Calculate Interest\n";
          cout <<"       6: Close an Account\n";
          cout <<"       7: Exit\n\n";
          cout <<"       Please select a choice : ";
          choice = getche();

                if (choice == '1')
                {
                       accountTransactions objAT;
                       objAT.new_account();
                }
                else
                if (choice == '2')
                {
                       accountTransactions objAT;
                       objAT.display_account();
                }
                else
                if (choice == '3')
                {
                       dispRecords newRec;
                       newRec.displayCustomers();
                }
                else
                if (choice == '4')
```

Hidden page

Hidden page

```
{
   int record ;
   record = getRecord(retrieve_AccNo) ;
   fstream filename ;
   filename.open("newrecords.dat", ios::in);
   filename.seekg(0,ios::end);
   int location;
   location = (record) * sizeof(dispRecords);
   filename.seekp(location);

   while (filename.read((char *) this, sizeof(dispRecords)))
   {
         if (retrieve_AccNo == AccountNumber)
         {
               cout <<"\n     ACCOUNT NO. : " <<AccountNumber ;
               cout <<"\n     Name     : "<<name ;
               cout <<"\n     Address : " <<address ;
               cout <<"\n     Balance : " <<intBalance ;
               break ;
         }
   }
   filename.close() ;
}

// getName() method returns the Account Holder's Name from the newrecords.dat file
char *dispRecords :: getName(int retrieve_AccNo)
{
   fstream filename;
   filename.open("newrecords.dat", ios::in);
   filename.seekg(0,ios::beg);
   char retrieve_CustName[30];

         while (filename.read((char *) this, sizeof(dispRecords)))
         {
               if (AccountNumber == retrieve_AccNo)
               {
                     strcpy(retrieve_CustName,name);
               }
         }
         filename.close();
         return retrieve_CustName;
}

// getAddress() method returns the Address of the Account Holder from the newrecords.dat
file
char *dispRecords :: getAddress(int retrieve_AccNo)
```

Hidden page

```
                {
                        count = 1;
                        break;
                }
        }
        filename.close();
        return count;
}

/* displayList() method displays the output of all the Accounts in a proper format
for the Choice 3*/
void dispRecords :: displayList()
{
        cout<<"
                                        \n" ;
        int day1, month1, year1 ;
        struct date dateval;
        getdate(&dateval);
        day1 = dateval.da_day ;
        month1 = dateval.da_mon ;
        year1 = dateval.da_year ;
        cout <<"\n Date: " <<day1 <<"/" <<month1 <<"/" <<year1<<"\n";
        cout<<setw(80)<<"————————————————————————————————————\n";
        cout<<setw(23)<<" ACCOUNT NO.";
        cout<<setw(23)<<" NAME OF PERSON";
        cout<<setw(23)<<" BALANCE\n";
        cout<<setw(80)<<"————————————————————————————————————\n";
}

// displayCustomers() method displays all the Account Holders/Customers from the
newrecords.dat file
void dispRecords :: displayCustomers(void)
{
        clrscr() ;
        int len1;
        int row=8, check ;
        fstream filename ;

        FILE * pFile;
        pFile = fopen("newrecords.dat","r");
        if (pFile == NULL)
        {
            cout<<"\n  No Account exists. Please go back to the previous menu. \n";
                getch();
                return ;
                //fclose (pFile);

        } else {
            displayList();
            filename.open("newrecords.dat", ios::in);
            filename.seekg(0,ios::beg);
            while (filename.read((char *) this, sizeof(dispRecords)))
            {
                  check = 0 ;

                  cout.fill(' ');
                  cout <<setw(20);
                  cout.setf(ios::right,ios::adjustfield);
                  cout<<AccountNumber;
                  cout.fill(' ');
                  cout <<setw(25);
                  cout.setf(ios::internal,ios::adjustfield);
                  cout<<name;

                  cout <<setw(23);
                  cout.setf(ios::right,ios::adjustfield);
                  cout<<intBalance<<"\n" ;
                  row++ ;
                  if (row == 23)
                  {
                        check = 1 ;
                        row = 8 ;
                        cout <<" \n\n Continue the application... \n";
                        getch() ;
                        clrscr() ;
                        displayList() ;
                  }
            }
      }
      filename.close() ;
      if (!check)
      {
            cout <<"\n\n Continue the application... \n";
            getch() ;
      }
}

// addDetails() method adds new records of Account Holders/Customers in the
newrecords.dat file
void dispRecords :: addDetails(int retrieve_AccNo, char retrieve_CustName[30],
char retrieve_Address[60], float iBalance)
{
      AccountNumber = retrieve_AccNo ;
      strcpy(name,retrieve_CustName) ;
      strcpy(address,retrieve_Address) ;
      intBalance = iBalance ;
```

Hidden page

```
    intBalance = iBalance ;
    int location ;
    location = (record-1) * sizeof(dispRecords) ;
    filename.seekp(location) ;
    filename.write((char *) this, sizeof(dispRecords)) ;
    filename.close() ;
}

// addDetails() method adds the details of a transaction in the transactions.dat file
void accountTransactions :: addDetails(int retrieve_AccNo, int day1, int month1, int
year1, char t_tran, char typeTransaction[10], float interest_accrued, float t_amount,
float iBalance)
{
    fstream filename ;
    filename.open("transactions.dat", ios::app) ;
    AccountNumber = retrieve_AccNo ;
    dday = day1 ;
    mmonth = month1 ;
    yyear = year1 ;
    transactions = t_tran ;
    strcpy(trantype,typeTransaction) ;
    intInterest = interest_accrued ;
    intAmount = t_amount ;
    intBalance = iBalance ;
    filename.write((char *) this, sizeof(accountTransactions)) ;
    filename.close();
}

// deleteAccount() method deletes the record of a transaction from the transactions.dat
file
void accountTransactions :: deleteAccount(int retrieve_AccNo)
{
    fstream filename ;
    filename.open("transactions.dat", ios::in) ;
    fstream temp ;
    temp.open("calculations.txt", ios::out) ;
    filename.seekg(0,ios::beg) ;
    while ( !filename.eof() )
    {
            filename.read((char *) this, sizeof(accountTransactions)) ;
            if ( filename.eof() )
                    break ;
            if ( AccountNumber != retrieve_AccNo )
                    temp.write((char *) this, sizeof(accountTransactions)) ;
    }
    filename.close() ;
    temp.close() ;
    filename.open("transactions.dat", ios::out) ;
    temp.open("calculations.txt", ios::in) ;
    temp.seekg(0,ios::beg) ;
    while ( !temp.eof() )
    {
            temp.read((char *) this, sizeof(accountTransactions)) ;
            if ( temp.eof() )
                    break ;
            filename.write((char *) this, sizeof(accountTransactions)) ;
    }
    filename.close() ;
    temp.close() ;
}

// new_account() method adds a new record in the newrecords file and transaction.dat
files(choice 1)
void accountTransactions :: new_account(void)
{
    char choice ;
    int i, check ;
    clrscr() ;
    dispRecords newRec ;
    cout <<"        Please press 0 to go back to previous menu. \n" ;
    cout<<"                                                    \n";
    cout<<"        —Open a New Bank Account—     \n";
    cout<<"        ********************   \n";
    int day1, month1, year1 ;
    struct date dateval;
    getdate(&dateval);
    day1 = dateval.da_day ;
    month1 = dateval.da_mon ;
    year1 = dateval.da_year ;
    int retrieve_AccNo ;
    retrieve_AccNo = newRec.lastAccount() ;
    retrieve_AccNo++ ;

    if (retrieve_AccNo == 1)
    {
            newRec.addDetails(retrieve_AccNo,"Ravi","Delhi",1.1)  ;
            newRec.deleteAccount(retrieve_AccNo)  ;
            addDetails(retrieve_AccNo,1,1,1997,'D',"default value",1.1,1.1,1.1) ;
            deleteAccount(retrieve_AccNo) ;
    }
    char retrieve_CustName[30], tran_acc[10], retrieve_Address[60] ;
    float t_bal, iBalance ;
    cout <<"        Date : "<<day1 <<"/" <<month1 <<"/" <<year1<<"\n" ;
    cout <<"        Account no. # " <<retrieve_AccNo;
```

Hidden page

```
            gets(chr_VerifyingPerson);
            if (chr_VerifyingPerson[0] == '0')
            {
                  cout<<"\n\t    Invalid Verifying Person Name.";
                  getch();
                  return;
            }
            strupr(chr_VerifyingPerson) ;
            if (strlen(chr_VerifyingPerson) < 1 || strlen(chr_VerifyingPerson) > 30)
            {
                  check = 0 ;
                  cout<<"\t\n    The Verifying Person's Name is either blank or
greater than 30 characters. Please try again.\n";
                  getch() ;
            }
      } while (!check) ;

      do
      {
            cout <<"\n  Please enter the Deposit Amount while opening a New Account : ";
            check = 1 ;
            gets(tran_acc) ;
            t_bal = atof(tran_acc) ;
            iBalance = t_bal ;
            if (strlen(tran_acc) < 1) {
                  cout<<"\n    Invalid Transaction value. Exiting from the current
Menu.\n ";
                  getch();
                  return ;
            }
            if (iBalance < 1000)
            {
                  check = 0 ;
                  cout<<"\t\n    The Minimum Deposit Amount should be Rs.1000. Please
try again. \n";
                  getch() ;
            }

      } while (!check) ;

      do
      {
            cout <<"\n     Do you want to save the record? (y/n) : " ;
            choice = getche() ;
            choice = toupper(choice) ;
```

```
    } while (choice != 'N' && choice != 'Y') ;
    if (choice == 'N' || choice == 'n')
    {
          cout<<"\n      The Customer Account is not created\n.
  Please continue with the application.\n";
          getch();
          return ;
    }
    float t_amount, interest_accrued ;
    t_amount = iBalance ;
    interest_accrued = 0.0 ;
    char t_tran, typeTransaction[10] ;
    t_tran = 'D' ;
    strcpy(typeTransaction," ") ;

    newRec.addDetails(retrieve_AccNo,retrieve_CustName,retrieve_Address,iBalance) ;
    addDetails(retrieve_AccNo,day1,month1,year1,t_tran,typeTransaction,
  interest_accrued,t_amount,iBalance);
    cout<<" \n\n          The New Account is successfully created.\n
  Please continue with the application.\n";
    getch();
  }

  // showAccount() method formats the display of the records from the transactions.dat
  file for a particular account(choice 2).
  void accountTransactions :: showAccount(int retrieve_AccNo)
  {
    cout<<"
                                                    \n";
    int day1, month1, year1 ;
    struct date dateval;
    getdate(&dateval);
    day1 = dateval.da_day ;
    month1 = dateval.da_mon ;
    year1 = dateval.da_year ;
    cout<<"Date: " <<day1 <<"/" <<month1 <<"/" <<year1<<"\n" ;
    cout <<"Account no. " <<retrieve_AccNo ;
    dispRecords newRec ;

    char retrieve_CustName[30] ;
    strcpy(retrieve_CustName,newRec.getName(retrieve_AccNo)) ;
    char retrieve_Address[60] ;
    strcpy(retrieve_Address,newRec.getAddress(retrieve_AccNo)) ;

    cout<<setw(25)<<"\n Account Holder's Name :  "<<retrieve_CustName;
    cout<<"\nAddress                :      "<<retrieve_Address<<"\n";
    cout<<setw(80)<<"\n———————————————————————————————————\n";
    cout<<setw(10)<<"Dated";
    cout<<setw(12)<<"Details";
    cout<<setw(12)<<"Deposited";
    cout<<setw(15)<<"Withdrawn";
    cout<<setw(12)<<"          ";
    cout<<setw(10)<<"Balance";
    cout<<setw(80)<<"\n―――――――――――――――――――――――――――――――――――\n";
}

// display_account() method displays records from the transactions.dat file
void accountTransactions :: display_account(void)
{
    clrscr() ;
    char t_acc[10] ;
    int tran_acc, retrieve_AccNo;
    dispRecords obj2;
    cout <<"        Press 0 to go back to previous menu.\n" ;
    cout <<"        Please enter Account No. you want to view :  " ;
    gets(t_acc);
    tran_acc = atoi(t_acc);       /* converting Account Number to integer value */
    retrieve_AccNo = tran_acc;
    if (retrieve_AccNo == 0){
            cout<<"\n      You have pressed 0 to exit. \n";
            getch();
            return ;
    }
    clrscr();
    dispRecords newRec;
    accountTransactions aa;
    int row=8, check ;
    fstream filename ;

    FILE * pFile;
    pFile = fopen("newrecords.dat","r");
    if (pFile == NULL)
    {
        cout<<"\n  No such Account Exists. Please create a New Account. \n";
            getch();
            return ;

    } else if (!newRec.accountExists(retrieve_AccNo)) {
            cout <<"\t\n   Account does not exist.\n";
            getch();
            return;
    } else {

            showAccount(retrieve_AccNo) ;
            filename.open("transactions.dat", ios::in);
```

```
// dateDiffer() method displays the difference between 2 dates.
int accountTransactions :: dateDiffer(int day1, int month1, int year1, int day2,
int month2, int year2)
{
   static int monthArr[] = {31,28,31,30,31,30,31,31,30,31,30,31};     //Array of
months for storing the no. of days in each array
   int days = 0 ;
   while (day1 != day2 || month1 != month2 || year1 != year2)
   {
         /* checking if the two dates in days,months and years differ and incrementing
the number of days.*/
         days++ ;
         day1++ ;
         if (day1 > monthArr[month1-1])
         {
               day1 = 1 ;
               month1++ ;
         }
         if (month1 > 12)
         {
               month1 = 1 ;
               year1++ ;
         }
   } return days ;
}

// getInterest() function calculates interest on the balance from the transaction.dat
file
float accountTransactions :: getInterest(int retrieve_AccNo, float iBalance)
{
   fstream filename ;
   filename.open("transactions.dat", ios::in);
   dispRecords newRec;
   filename.seekg(0,ios::beg) ;
   int day1, month1, year1, month_day;
   while (filename.read((char *) this, sizeof(accountTransactions)))
   {
         if (AccountNumber == retrieve_AccNo)
         {
               day1 = dday ;
               month1 = mmonth ;
               year1 = yyear ;
               iBalance = newRec.getBalance(retrieve_AccNo);
               break ;
         }
   }
   int day2, month2, year2;
   struct date dateval;
   getdate(&dateval);
   day2 = dateval.da_day;
   month2 = dateval.da_mon;
   year2 = dateval.da_year;
   float interest_accrued=0.0;
   int yeardiff = year2 - year1;

   if ((year2<year1) || (year2==year1 && month2<month1) || (year2==year1 &&
month2==month1 && day2<day1)) {

          return interest_accrued;
   }
   month_day = dateDiffer(day1,month1,year1,day2,month2,year2);
   int months;
   if (month_day >= 30)
   {
          months = month_day/30;
   } else {
          months = month_day/30;
   }
          if(interest_accrued == 0 && yeardiff == 1) {
                 interest_accrued = ((iBalance*0.5)/100) * (months);
          } else if (yeardiff > 1 && yeardiff < 25 && interest_accrued == 0) {
                        interest_accrued = ((iBalance*0.5)/100) * (months);
          } else {
                 interest_accrued = 0;
          }

   filename.close();
   return interest_accrued;
}

/*Method for generating Interest and updation of the Balance and addDetails
methods.(Choice 5)*/
void accountTransactions :: showInterest(void)
{
   clrscr();
   char t_acc[10];
   int tran_acc, retrieve_AccNo, check;

   cout <<strupr("\n      Important Information: Interest should be generated only\n
once in a Year.\n\n\t If you have already generated interest for an Account,\n\t
please ignore that Account.\n\t Thank you.\n");
   cout <<"\n      Press 0 to go back to previous menu.\n" ;
   cout <<"\n      To view the transaction of the Account, please enter it: " ;
   gets(t_acc) ;
```

Hidden page

```
/* This method does all the Deposit/Withdrawal transactions in the transaction.dat
file(Choice 4)*/
void accountTransactions :: transaction(void)
{
   clrscr();
   char t_acc[10];
   int tran_acc, retrieve_AccNo, check;
   cout <<"       Press 0 to go back to previous menu.\n" ;
   cout <<"       To view the transaction of the Account, please enter it: " ;
   gets(t_acc) ;
   tran_acc = atoi(t_acc) ;
   retrieve_AccNo = tran_acc ;
   if (retrieve_AccNo == 0)
         return ;
   clrscr() ;
   dispRecords newRec ;
   if  (!newRec.accountExists(retrieve_AccNo))
   {
         cout <<"\t\n   Account does not exist.\n";
         getch();
         return;
   }
   cout <<"       Press 0 to go back to previous menu.\n";
   cout<<"                                                  \n";
   cout<<"\n      —Make correct entry for the Transaction below—  \n";
   cout<<" ********************************************\n";
   int day1, month1, year1;
   struct date dateval;
   getdate(&dateval);
   day1 = dateval.da_day;
   month1 = dateval.da_mon;
   year1 = dateval.da_year;
   cout <<"       Date : "<<day1 <<"/" <<month1 <<"/" <<year1<<"\n";
   cout <<"       Account no. " <<retrieve_AccNo<<"\n";
   char retrieve_CustName[30] ;
   char retrieve_Address[60] ;
   float iBalance;
   float interest_accrued = 0.0;
   strcpy(retrieve_CustName,newRec.getName(retrieve_AccNo)) ;
   strcpy(retrieve_Address,newRec.getAddress(retrieve_AccNo)) ;
   iBalance = newRec.getBalance(retrieve_AccNo);

   cout <<"       Customer Name    : " <<retrieve_CustName;
   cout <<"\n     Customer Address: " <<retrieve_Address ;
   cout <<"\n     Bank Balance: " <<iBalance ;
   char tranDetails, typeTransaction[10], tm[10] ;
   float t_amount, t_amt ;
```

```
    do
    {
            cout <<"\n  Please enter D for Deposit or W for Withdrawal of Amount : " ;
            tranDetails = getche() ;
            if(tranDetails == '0') {
                    cout<<"\n\n    You have pressed 0 to Exit.";
                    getch();
                    return;
            }
            tranDetails = toupper(tranDetails) ;
    } while (tranDetails != 'W' && tranDetails != 'D') ;

    do
    {
            cout <<"\n     The Transaction type is either Cash or Cheque..\n" ;
            check = 1 ;
            cout <<"       (Cash/Cheque) : " ;
            gets(typeTransaction) ;
            strupr(typeTransaction);
            if(typeTransaction[0] == '0') {
                    cout<<"\n\n    You have pressed 0 to Exit.";
                    getch();
                    return;
            }
            if (strlen(typeTransaction) < 1 || (strcmp(typeTransaction,"CASH") &&
  strcmp(typeTransaction,"CHEQUE")) )
            {
                    check = 0 ;
                    cout<<"\n      The Transaction is invalid. Please enter either
  Cash or Cheque. \n" ;
                    getch() ;
            }

    } while (!check);

    do
    {
            cout <<"\n     Please enter the Transaction Amount :  \n";
            check = 1 ;
            cout <<"       Amount : Rs. ";
            gets(tm) ;
            t_amt = atof(tm) ;
            t_amount = t_amt ;

            if (t_amount < 1 || (tranDetails == 'W' && t_amount > iBalance) )
```

Hidden page

```
    tran_acc = atoi(t_acc) ; /* changing account no. to integer type. */
    retrieve_AccNo = tran_acc ;
    clrscr() ;
    dispRecords newRec ;
    if (!newRec.accountExists(retrieve_AccNo))
    {
            cout <<"\t\n You have entered an invalid Account or it does not exist.\n";
            cout <<" Please try again.\n";
            getch();
            return ;
    }
    cout <<"\n     Press 0 to go back to previous menu\n" ;
    cout<<"\n      Closing this Account.\n";
    cout<<"**********************************\n\n";
    int day1, month1, year1 ;
    struct date dateval;
    getdate(&dateval);
    day1 = dateval.da_day ;
    month1 = dateval.da_mon ;
    year1 = dateval.da_year ;

    cout <<"Date: "<<day1 <<"/" <<month1 <<"/" <<year1<<"\n";
    char choice;
    newRec.display(retrieve_AccNo); /*Displaying the Account Details on the basis of
  the retrieved Account Number*/

    do
    {
            cout <<"\n     Are you sure you want to close this Account? (y/n): ";
            choice = getche();
            choice = toupper(choice) ;
    } while (choice != 'N' && choice != 'Y');

    if (choice == 'N' || choice == 'n') {
            cout<<"\n      The Account is not closed.\n";
            getch();
            return;
    }
    newRec.deleteAccount(retrieve_AccNo);
    deleteAccount(retrieve_AccNo);
    cout <<"\t\n\n Record Deleted Successfully.\n";
    cout <<"       Please continue with the application....\n";
    getch();
  }

  /* The Login method checks for the username and the password for accessing the
  Banking Application*/
```

```
int login (void)
{
char username[9],ch;
char username1[]="banking";
int i=0;
char a,b[9],pass[]="tatahill";
cout<<"\n\n";
cout<<"\n\t    Login to the Banking Application.\n";
cout<<"\t      ***********************************\n";
cout<<"\n\n\tPlease enter Username    :       ";
cin >> username;
cout<<"\n\n\tPlease enter Password to authenticate yourself :       ";
fflush(stdin);
        do
                        {
                        ch=getch();
                        if(isalnum(ch))
                                {
                                b[i]=ch;
                                cout<<"*";
                                i++;
                                }
                        else
                                if(ch=='\r')
                                        b[i]='\0';
                                else if(ch=='\b')
                                        {
                                        i--;
                                        cout<<"\b\b";
                                        }
                        }
                while(ch!='\r');

b[i]='\0';
fflush(stdin);

        if((strcmp(b,pass)==0)&&(strcmp(username1,username)==0))
        {
                        cout<<"\n\n\t  You have entered successfully\n\n";
                        return(1);
        }
        else
        {
                        cout<<"\t\n\n          Incorrect Username or Password.";
                        cout<<"\n";
                        return(0);
        }
```

Hidden page

Appendix B

Executing Turbo C++

B.1 Introduction

All programs in this book were developed and run under Turbo C++ compiler Version 3.0, in an MS-DOS environment on an IBM PC compatible computer. We shall discuss briefly, in this Appendix, the creation and execution of C++ programs under Turbo C++ system.

B.2 Creation and Execution of Programs

Executing a computer program written in any high-level language involves several steps, as listed below:

1. Develop the program (source code).
2. Select a suitable file name under which you would like to store the program.
3. Create the program in the computer and save it under the filename you have decided. This file is known as *source code file.*
4. Compile the source code. The file containing the translated code is called *object code file*. If there are any errors, debug them and compile the program again.
5. Link the object code with other library code that are required for execution. The resulting code is called the *executable code*. If there are errors in linking, correct them compile the program again.
6. Run the executable code and obtain the results, if there are no errors.
7. Debug the program, if errors are found in the output.
8. Go to Step 4 and repeat the process again.

These steps are illustrated in Fig. B.1. The exact steps depend upon the program environment and the compiler used. But, they will resemble the steps described above.

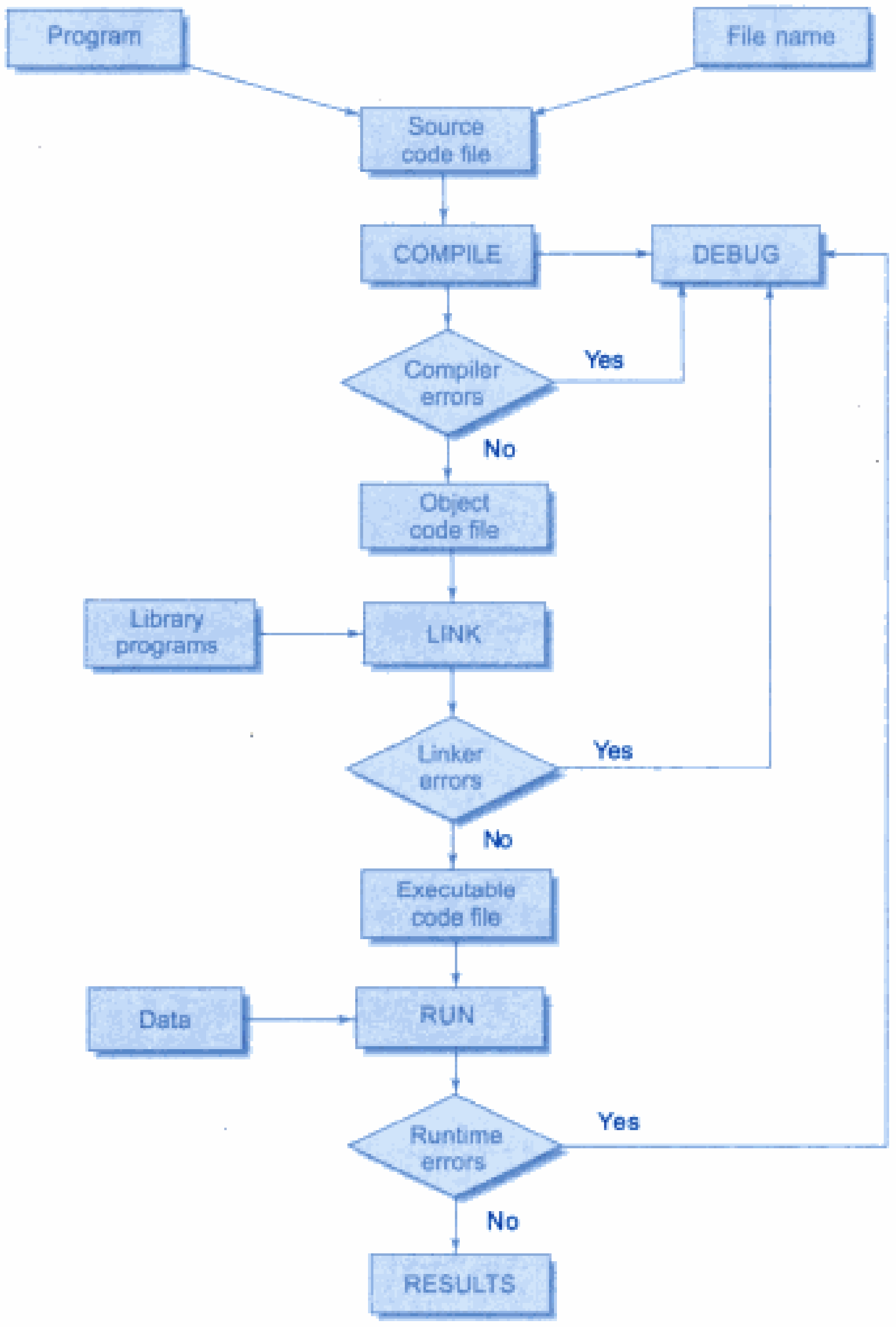


Fig. B.1 ⇔ *Program development and execution*

Turbo C++ and Borland C++ are the two most popular C++ compilers. They provide ideal platforms for learning and developing C++ programs. In general, both Turbo C++ and Borland C++ work the same way, except some additional features supported by Borland C++ which are outside the scope our discussions. Therefore, whatever we discuss here about Turbo C++ applies to Borland C++ as well.

B.3 Turbo C++

Turbo C++ provides a powerful environment called ***Integrated Development Environment*** **(IDE)** for creating and executing a program. The IDE is completely menu-driven and allows the user to create, edit, compile and run programs using what are known as ***dialogue boxes.*** These operations are controlled by single keystrokes and easy-to-use menus.

We first use the editor to create the source code file, then compile, link and finally run it. Turbo C++ provides error messages, in case errors are detected. We have to correct the errors and compile the program again.

B.4 IDE Screen

It is important to be familiar with the details of the IDE screen that will be extensively used in the program development and execution. When we invoke the Turbo C++, the IDE screen will be displayed as shown in Fig. B.2. As seen from the figure, this screen contains four parts:

- Main menu (top line)
- Editor window
- Message window
- Status line (bottom line)

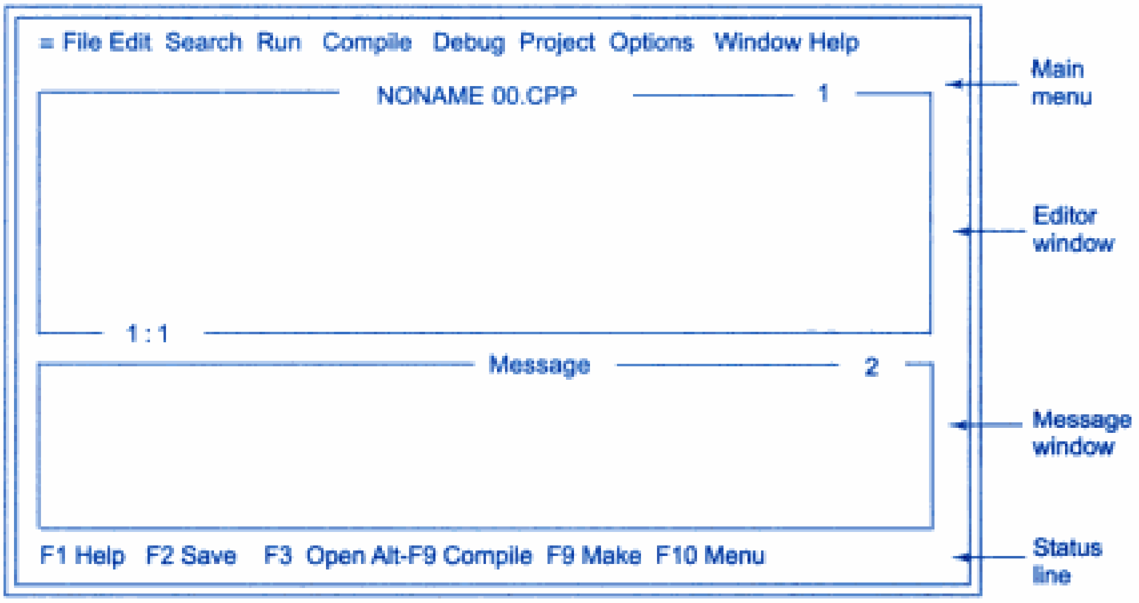

≡ File Edit Search Run Compile Debug Project Options Window Help
NONAME 00.CPP
1
1 : 1
Message
2
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu
Main menu
Editor window
Message window
Status line

Fig. B.2 ⇔ IDE opening screen

Main Menu

The *main menu* lists a number of items that are required for the program development and execution. They are summarized in Table B.1.

Table B.1 *Main menu items*

Item	*Options*
_	Displays the version number, clears or restores the screen, and execute various utility programmes supplied with Turbo C++
File	Loads and saves files, handless directories invokes DOS, and exists Turbo C+
Edit	Performs various editing functions
Search	Performs various text searches and replacements
Run	Complies, links and runs the program currently loaded in the environment
Compile	Compiles the program currently in the environment
Debug	Sets various debugger options, including setting break points
Projects	Manages multifile projects
Options	Sets various compiler, linker, and environmental options
Window	Controls the way various windows are displayed
Help	Activates the context–sensitive Help system

The *main menu* can be activated by pressing the F10 key. When we select an item on the main menu, a *pull-down menu*, containing various options, is displayed. This allows us to select an action that relates to the main menu item.

Editor Window

The *editor window* is the place for creating the source code of C++ programs. This window is named NONAME00.CPP. This is the temporary name given to a file which can be changed while we save the file.

Message Window

The other window on the screen is called the *message window* where various messages are displayed. The messages may be compiler and linker messages and error messages generated by the compiler.

Status Line

The *status line* which is displayed at the bottom of the screen gives the status of the current activity on the screen. For example, when we are working with FILE option of main menu, the status line displays the following:

F1 Help | Locate and open a file

B.5 Invoking Turbo C++

Assuming that you have installed the Turbo C++ compiler correctly, go to the directory in which you want to work. Then enter TC at the DOS system prompt:

```
C:>TC
```

and press RETURN. This will place you into the IDE screen as shown in Fig. B.2. Now, you are ready to create your program.

B.6 Creating Source Code File

Once you are in the IDE screen, it is simple to create and save a program. The F10 key will take you to main menu and then move the cursor to *File*. This will display the file dialogue window containing various options for file operations as shown in Fig. B.3. The options include, among others, opening an existing file, creating a new file and saving the new file.

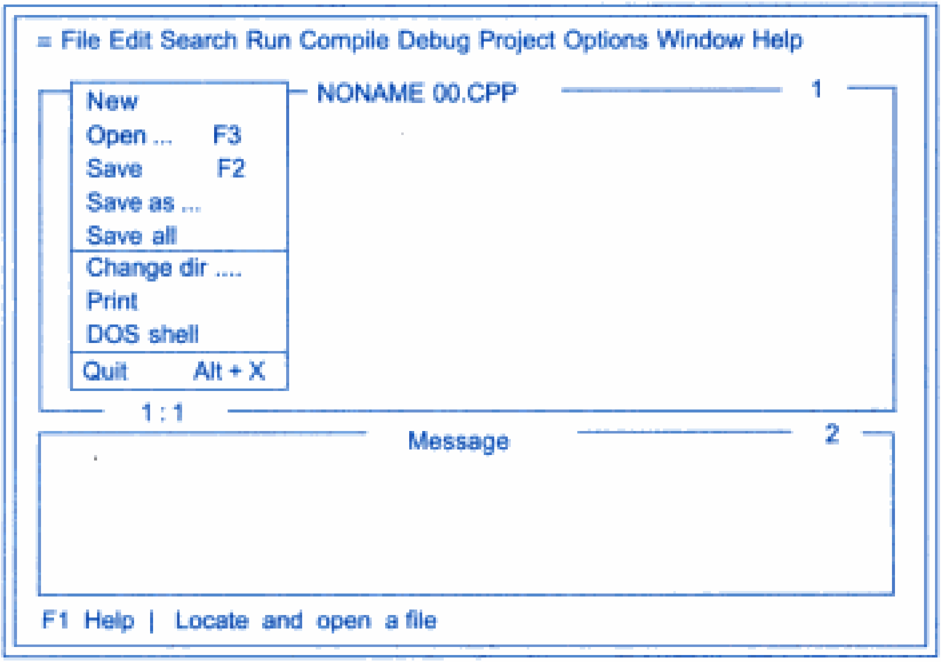
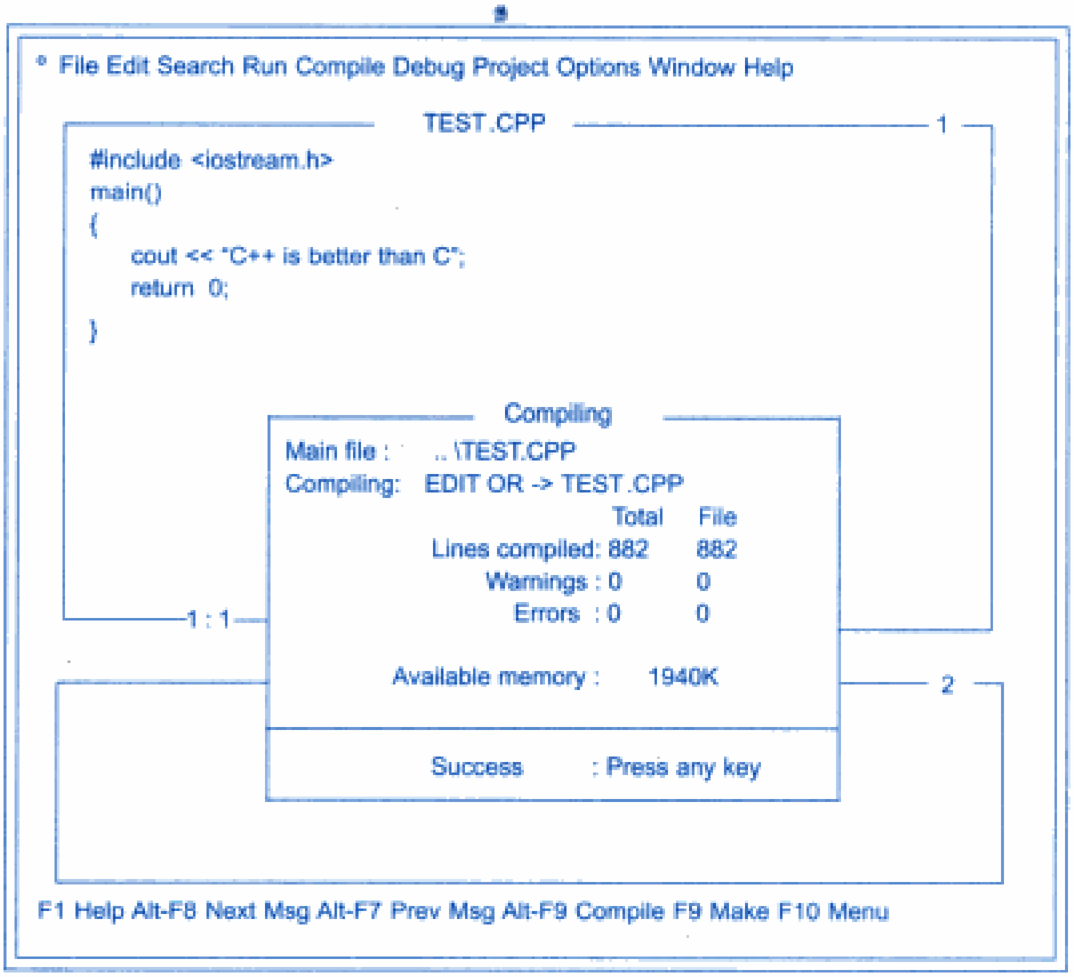


Fig. B.3 ⇔ File dialogue window

Since you want to create a new file, move the cursor to **New** option. This opens up a blank window called *editing window* and places the cursor inside this window. Now the system is ready to receive the program statements as shown in Fig. B.4.

Hidden page


File Edit Search Run Compile Debug Project Options Window Help
TEST.CPP
1
1 : 1
2
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include <iostream.h>
main()
{
     cout << "C++ is better than C";
     return  0;
}
```

```
                    Compiling
Main file :      .. \TEST.CPP
Compiling:    EDIT OR -> TEST.CPP
                            Total     File
           Lines compiled: 882       882
                Warnings : 0         0
                   Errors  : 0         0

        Available memory :      1940K

              Success        : Press any key
```

Fig. B.7 ⇔ Compilation window

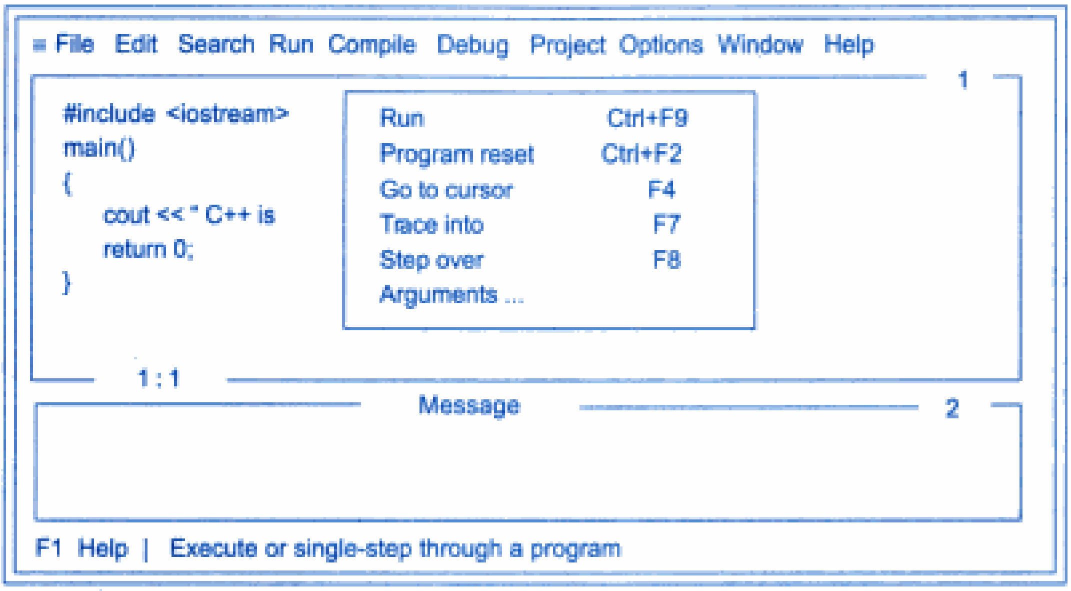
B.9 Running the Program

You have reached successfully the final stage of your excitement. Now, select the **Run** from the main menu and again **Run** from the run *dialogue window* (See Fig. B.8). You will see the screen flicker briefly. Surprisingly, no output is displayed. Where has the output gone? It has gone to a place known as *user screen*.

In order to see the user screen, select **window** from the main menu and then select *user screen* from the window dialogue menu (See Fig.B.9). The IDE screen will disappear and the user screen is displayed containing output of the program **test.cpp** as follows:

```
C > TC
```

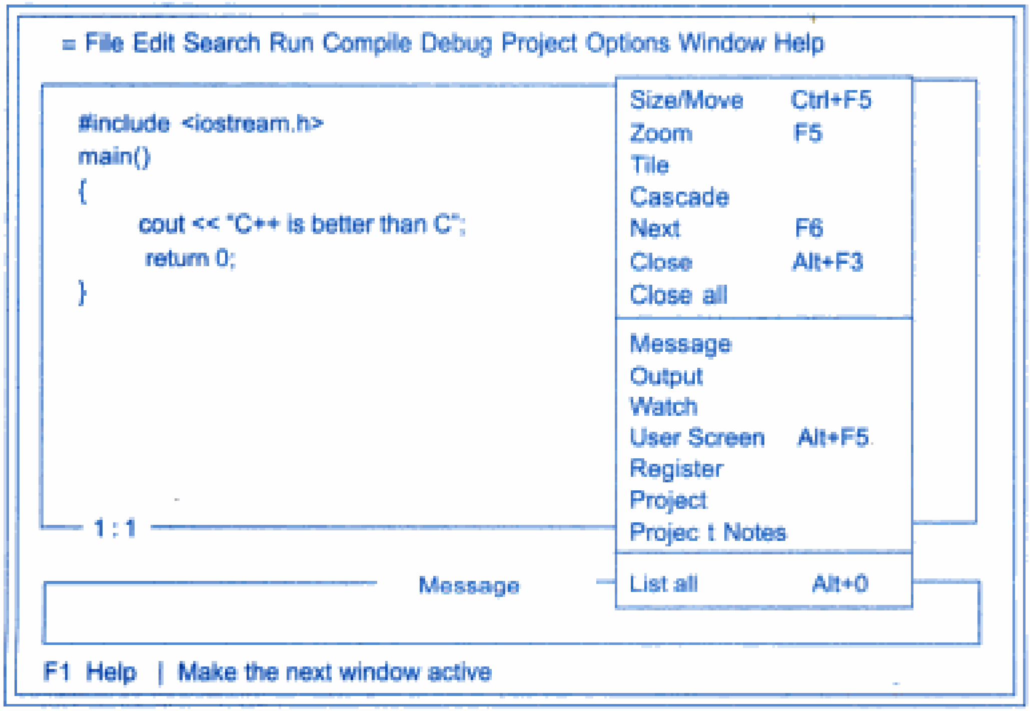
Note that, at this point, you are outside the IDE. To return to IDE, press RETURN key.



```
#include <iostream>
main()
{
     cout << " C++ is
     return 0;
}
```

F1 Help | Execute or single-step through a program

Fig. B.8 ⇔ *Run dialogue menu*



```
#include <iostream.h>
main()
{
        cout << "C++ is better than C";
        return 0;
}
```

F1 Help | Make the next window active

Fig. B.9 ⇔ *Window dialogue menu*

B.10 Managing Errors

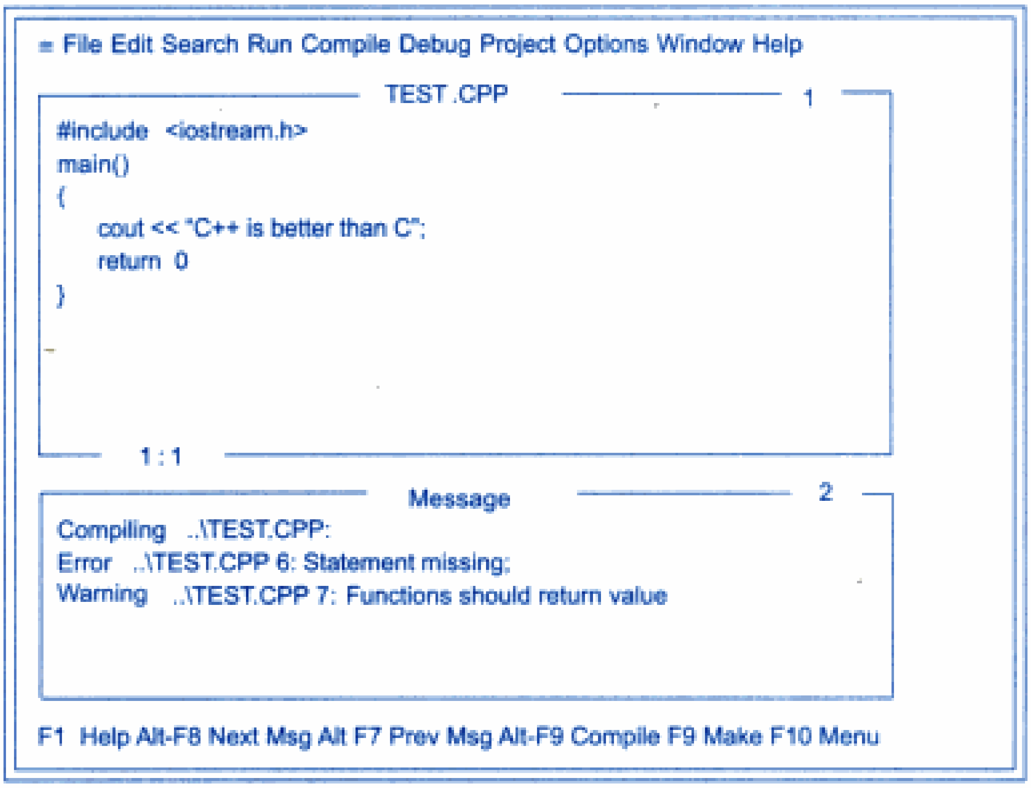
It is rare that a program runs successfully the first time itself. It is common to make some syntax errors while preparing the program or during typing. Fortunately, all such errors are detected by the compiler or linker.

Compiler Errors

All syntax errors will be detected by the compiler. For example, if you have missed the semicolon at the end of the **return** statement in **test.cpp** program, the following message will be displayed in the message window.

```
Error...\TEST.CPP 6 Statement missing;
Warning...\TEST.CPP 7: Function should return a value
```

The number 6 is the possible line in the program where the error has occurred. The screen now will look like the one in Fig. B.10.


≡ File Edit Search Run Compile Debug Project Options Window Help
TEST.CPP
1
1 : 1
Message
2
F1 Help Alt-F8 Next Msg Alt F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include <iostream.h>
main()
{
     cout << "C++ is better than C";
     return 0
}
```

```
Compiling ..\TEST.CPP:
Error ..\TEST.CPP 6: Statement missing;
Warning ..\TEST.CPP 7: Functions should return value
```

Fig. B.10 ⇔ Display of error message

Press ENTER key to go to **Edit** window that contains your program. Correct the errors and then compile and run the program again. Hopefully, you will obtain the desired results.

Linker Errors

It is also possible to have errors during the linking process. For instance, you may not have included the file *iostream.h*. The program will compile correctly, but will fail to link. It will display an error message in the *linking window*. Press any key to see the message in the message window.

Run-time Errors

Remember compiling and linking successfully do not always guaranty the correct results. Sometimes, the results may be wrong due logical errors or due to errors such as stack overflow. System might display the errors such as *null pointer assignment*. You must consult the manual for the meaning of such errors and modify the program accordingly.

B.11 Handling an Existing File

After saving your file to disk, your file has become a part of the list of files stored in the disk. How do we retrieve such files and execute the programs written to them? You can do this in two ways:

1. Under DOS prompt
2. Under IDE

Under DOS prompt, you can invoke as follows:

```
C > TC  TEST.CPP
```

Remember to type the complete and correct name of the file with **.cpp** extension. This command first brings Turbo C++ IDE and then loads **edit window** containing the file **test.cpp**.

If you are working under IDE, then select **open** option from the *file menu*. This will prompt you for a file name and then loads the file as you respond with the correct file name. Now you can edit the program, compile it and execute it as before.

B.12 Some Shortcuts

It is possible to combine the two steps of compiling and linking into one. This can be achieved by selecting **Make EXE file** from the compile dialogue window.

We can shorten the process by combining the execution step as well with the above step. In this case, we must select **Run** option from the run dialogue window. This causes the program to be compiled, linked and executed.

Many common operations can be activated directly without going through the main menu, again and again. Turbo C++ supports what are known as *hot keys* to provide these shortcuts. A list of hot keys and their functions are given Table B.2. We can use them whenever necessary.

Hot Key	*Meaning*
F1	Activates the online Help system
F2	Saves the file currently being edited
F3	Loads a file
F4	Executives the program unit the cursor is reached
F5	Zooms the active window
F6	Switches between windows
F7	Traces program; skips function calls
F8	Traces program; skips function calls
F9	Compiles and links programs
F10	Activates the main menu
ALT-O	Lists open windows
ALT-n	Activates window n (n must be 1 through 9)
ALT-F1	Shows the previous help screen
ALT-F3	Deletes the active window
ALT-F4	Opens an Inspector window
ALT-F5	Opens an Inspector window
ALT-F7	Previous error
ALT-F8	Next error
ALT-F9	Compiles file to .OBJ
ALT-SPACEBAR	Activates the main menu
ALT-C	Activates the Compile menu
ALT-D	Activates the Debug menu
ALT-E	Activates the Edit menu
ALT-F	Activates the File menu
ALT-H	Activates the Help menu
ALT-O	Activates the Options menu
ALT-P	Activates the Project menu
ALT-R	Activates the Run menu
ALT-S	Activates the Run menu
ALT-W	Activates the Window menu

(Contd)

Hidden page

Appendix C

Executing C++ Under Windows

C.1 Introduction

C++ is one of the most popular languages due to its power and portability. It is available for different operating systems such as DOS, OS/2, UNIX, Windows and many others. C++ programs when implemented under Windows are called Visual C++ programs. Therefore, there is no difference between C++ and Visual C++ programs in terms of programming but the difference lies in terms of implementation.

A C++ compiler designed for implementation under Windows is known as Visual C++. A C++ program running under MS-DOS will also run successfully under Windows. This is because, the rules of programming are the same; only the environment of implementation is different and is shown in Fig. C.1.

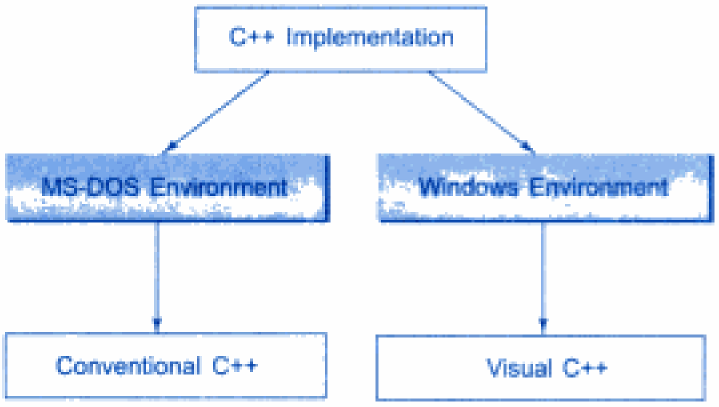


Fig. C.1 ⇔ *C++ Implementation environments*

A C++ programmer can easily become a Visual C++ programmer if he knows how to use the implementation tools of his Visual C++ system. In this Appendix, we introduce the features of Microsoft Visual C++ and discuss how to create, compile and execute C++ programs under Windows.

The Microsoft Corporation has introduced a Windows based C++ development environment named as **Microsoft Visual C++** (MSVC). This development environment integrates a set of tools that enable a programmer to create and run C++ programs with ease and style. Microsoft calls this integrated development environment (IDE) as **Visual Workbench. Microsoft Visual Studio**, a product sold by Microsoft Corporation, also includes Visual C++, in addition to other tools like Visual Basic, Visual J++, Visual Foxpro, etc.

C.2 The Visual Workbench

It is important to be familiar with the Visual Workbench that will be extensively used in the program development. The Visual Workbench is a visual user interface designed to help implement C++ programs. This contains various tools that are required for creating, editing, compiling, linking and running of C++ programs under Windows. These tools include File, Edit, Search, Project, Resource, Debug, Tools, Window and Help.

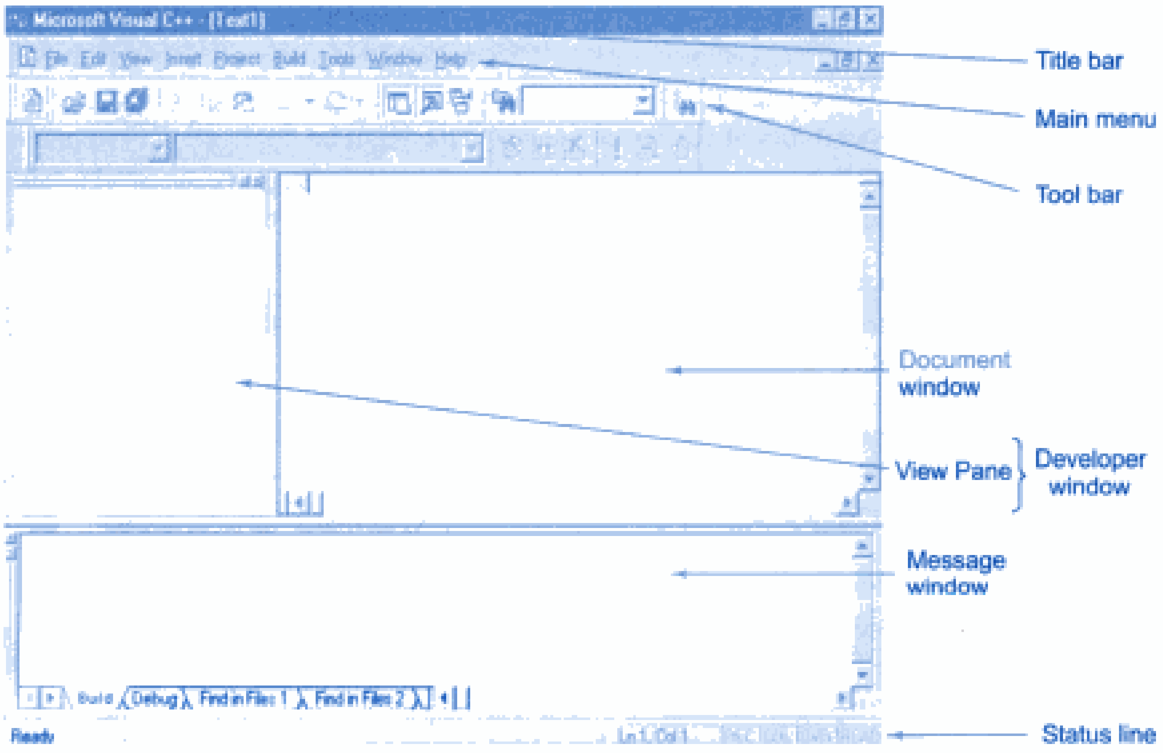

Microsoft Visual C++ - [Text1]
File Edit View Insert Project Build Tools Window Help
Build Debug Find in Files 1 Find in Files 2
Ready
Ln 1, Col 1
Title bar
Main menu
Tool bar
Document window
View Pane
Developer window
Message window
Status line

Fig. C.2 ⇔ *Visual workbench opening screen*

When we invoke the Microsoft Visual C++ (Version 6.0), the initial screen of the Visual Workbench will be displayed as shown in Fig. C.2. As seen from the figure, this screen contains five parts: 1) Title bar 2) Main menu 3) Tool bar 4) Developer window 5) Status line.

Main Menu

The main menu lists a number of items that are required for program development and execution. They are summarized in Table C.1.

Table C.1 *Main menu of visual workbench*

Item	*Functions/Options*
File	Creates a new file or opens an existing file for editing. Closes and saves files. Exits the Visual Workbench.
Edit	Performs various editing functions, such as searching, deleting, copying, cutting and pasting.
View	Enable different views of screen, output, workspace.
Insert	Insertion of Graphics resources like pictures, icons, HTML, etc. can be done.
Project	Sets up and edits a project (a list of files).
Build	Compiles the source code in the active window. Builds an executable file. Detects errors.
Tools	Customizes the environment, the editors and the debugger.
Window	Controls the visibility of various Windows involved in an application development.
Help	Provides help about using Visual C++ through Microsoft Developer Network Library (MSDN Library). Online help also can be received provided an Internet connection.

Once a main menu item is selected, a pull-down menu, containing various options, is displayed. This allows us to select an action/command that relates to the main menu item.

It is likely that an option in the pull-down menu is grayed. This means that the particular option is currently not available or not valid. For example, the Save option in the File menu will be grayed if the workspace is empty.

Some options are followed by three periods (...). Such an option, when selected, will display a submenu known as dialog box suggesting that some more input is required for that option to get implemented. Options followed by the symbol ▶ means we have to select a choice from the list.

Tool Bar

The tool bar resides just below the main menu. This provides a shortcut access to many of the main menu's options with a single mouse click. Figure. C.3 shows some important tool bar commands that can be used from anywhere within the Workbench. Several tool bars like Standard, Build, Edit, Wizard Bar, etc. are available which can be enabled/disabled from the screen using Tools/Customize option.

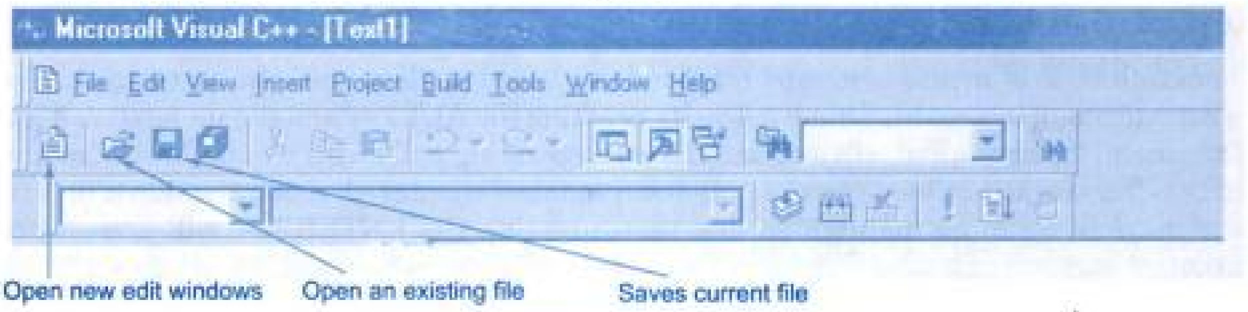

Microsoft Visual C++ - [Text1]
File Edit View Insert Project Build Tools Window Help
Open new edit windows
Open an existing file
Saves current file

Fig. C.3 ⇔ *Tool bar actions*

Developer Window

Just below the tool bar is the developer window. It is initially divided into three parts as shown in Fig. C.2.

- View Pane (on the left)
- Document window (on the right)
- Message window (at the bottom)

The view pane has three tabs for ClassView, FileView and InfoView. Once we have a project going, the ClassView will show us the class hierarchy and the FileView will show us the files used. InfoView will allow us to navigate through the documentation.

The document window, also known as workspace, is the place where we enter or display our programs. The message window displays messages such as warnings and errors when we compile the programs.

Accessing Menu Items

Before we proceed further, it is important to know how to access the menu items. There are two ways of accomplishing this:

1. Using the mouse
2. Using the keyboard

Mouse Actions

Using the mouse for accessing an item is the most common approach in Windows programming. We can perform the following actions with the mouse:

- Move the mouse pointer to a desired location by moving the mouse without pressing any button.
- Click the left mouse button when the pointer is over the preferred option.

Keyboard Actions

Though the use of mouse is a must for Windows-based applications, the accessing can also be done through keyboard. Simultaneously pressing the ALT key and the underscored letter of the menu item required will activate the corresponding pull-down menu. The underscored letter is known as hot key. Once a pull-down menu is displayed, using the down/up arrow keys an option can be highlighted and then pressing the ENTER key will activate that option.

Some of the options in a pull-down menu can be directly activated by using their hot key combinations shown against these options. For example, Ctrl+N is the hot key combination for the New option in the File menu. Similarly by pressing Ctrl+S, a file can be saved without using pull-down menu. This shortcut approach can be used from anywhere within the Visual Workbench.

C.3 Implementing Visual C++ Programs

Developing and implementing a computer program written in any high-level language involves several steps already described in Appendix B.

C.4 Creating a Source Code File

When you have installed the Microsoft Visual C++ compiler correctly, you can start the Visual Workbench from Microsoft Windows. To start the Visual Workbench, simply select the Visual C++ icon from the Programs group and click on it. This will bring up the Visual Workbench screen as shown in Fig. C.2. Once you are in the Visual Workbench screen, it is simple to create and save a program.

Entering the Program

The first thing you need to do before entering a program is to open a new file. Select the File menu from the main menu. This will display a pull-down file menu as shown in Fig. C.4. The options include, among others, opening an existing file, creating a new file and saving the new file.

Since, you want to create a new file, choose New ... option which will bring up the New dialog box as shown in Fig. C.5 displaying a list of different types of programming files.

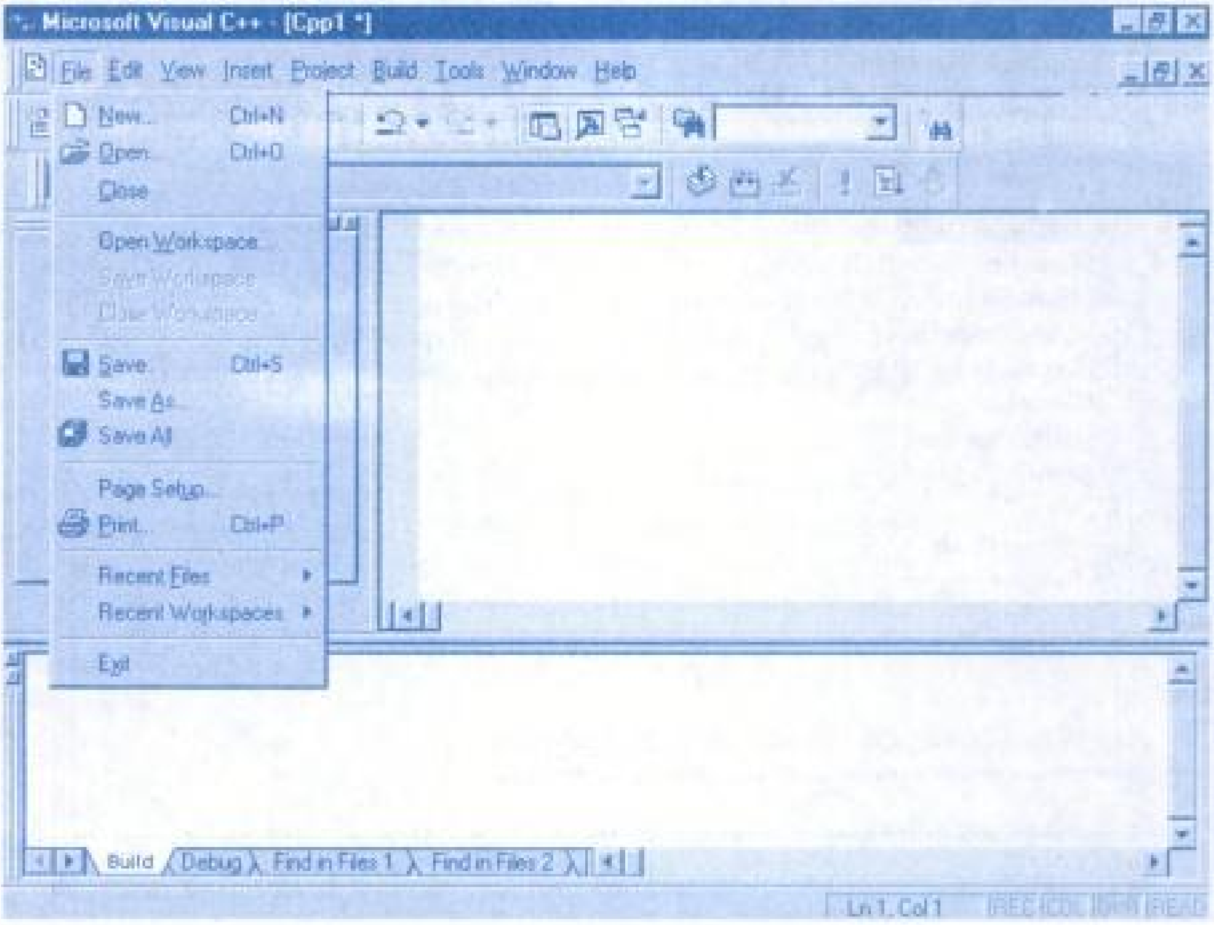


Fig. C.4 ⇔ *Visual C++ Workbench file menu*

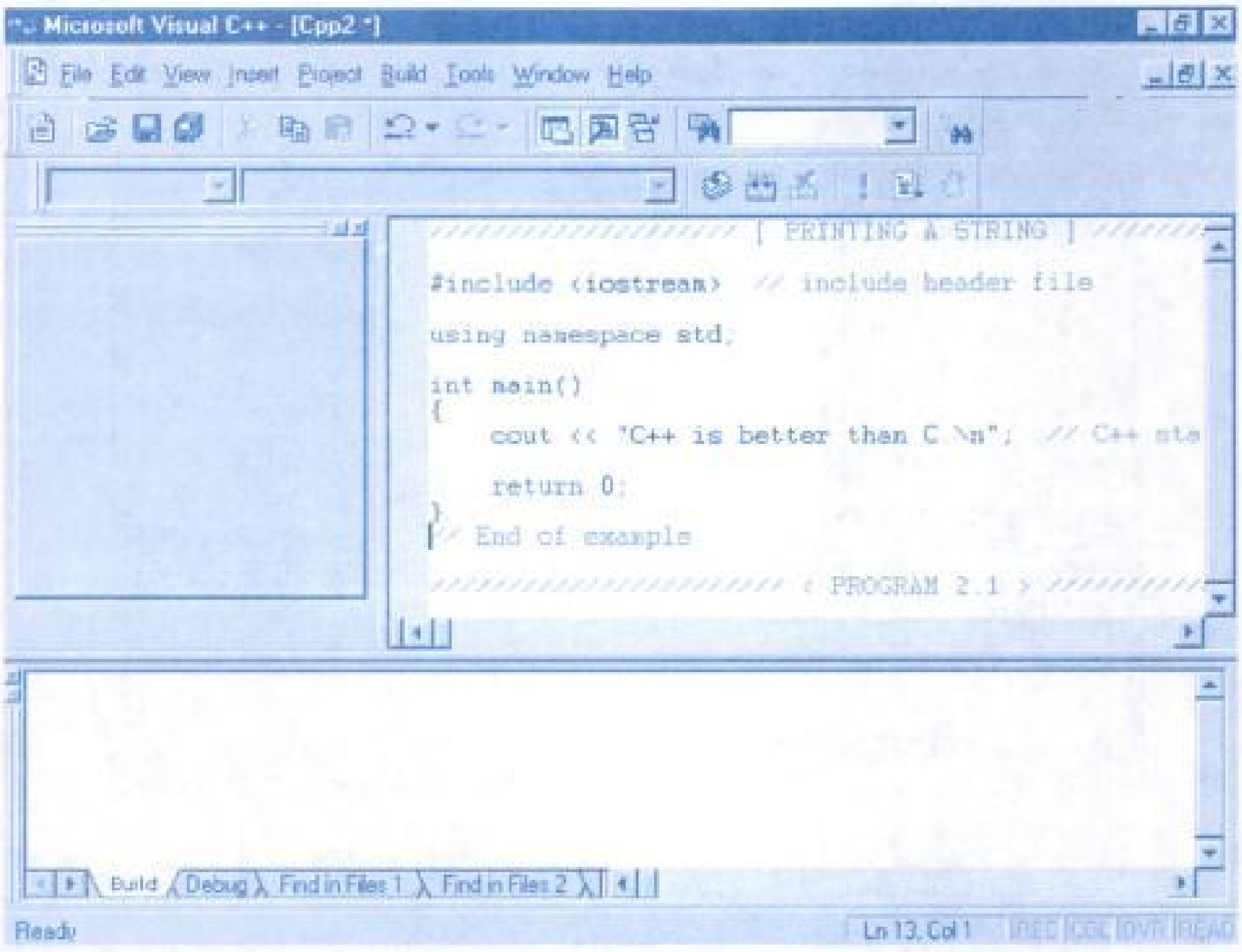
For entering a new program, select File/C++ Source File option and then click on the OK button. This opens up a blank window (similar to Fig. C.2) with the window title as 'Microsoft Visual C++ - [CPP]' and places the cursor inside this edit window. Now the system is ready to receive the program statements as shown in Fig. C.6.

Saving the Program

Once the typing is completed, you are ready to execute the program. Although a program can be compiled and run before it is saved, it is always advisable to save the program in a file before compilation. You can do so by doing one of the following:

1. Using File/Save command
2. Pressing the Ctrl+S hot key combination
3. Clicking on the third button from left on the toolbar.

Hidden page


Microsoft Visual C++ - [Cpp2 *]
File Edit View Insert Project Build Tools Window Help
Build Debug Find in Files 1 Find in Files 2
Ready
Ln 13, Col 1 REC COL OVR READ

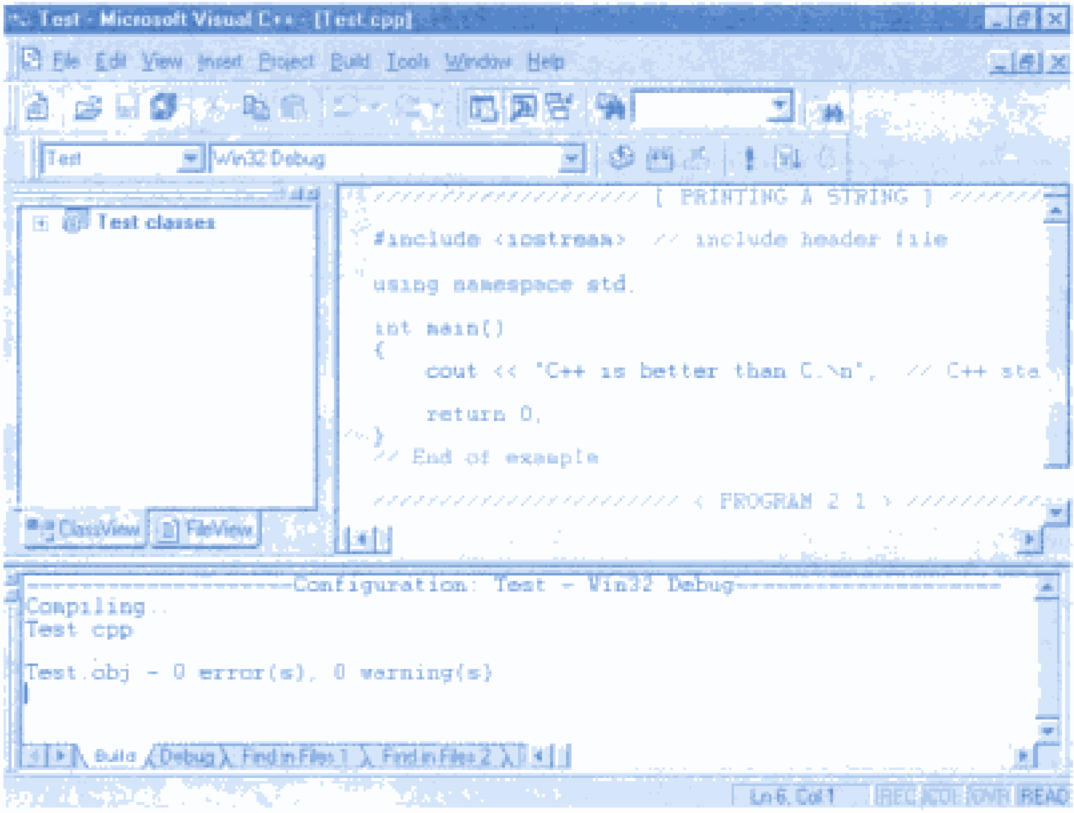
```
//////////////////////// [ PRINTING A STRING ] ////////
#include <iostream>  // include header file
using namespace std;
int main()
{
    cout << "C++ is better than C \n";  // C++ sta
    return 0;
}
// End of example
///////////////////////// < PROGRAM 2.1 > //////////
```

Fig. C.6 ⇔ *Edit window with the sample program*

Table C.2 *Three ways of compiling*

Command	***Action***
Build/Compile	Compiles a single program file. The result is an object file. This option is used when we want to check a particular file for syntax errors. Note that it does not link and therefore does not produce any executable file.
Build/Build	Compiles all the modified/new source files and then links all the object files to create a new executable file. When we are working on a project, we usually use this command. That is because we may change a few things here and there and want to compile only those modified programs.
Build/Rebuild All	Compiles all the files in a project and links them together to create an executable file. This command is usually used when we want to make sure that everything in the project has been built again.

The compile option in the Build menu when selected will compile the source code into an executable code if there is no errors or warnings as shown in Fig. C.7.



```
/////////////////////// [ PRINTING A STRING ] ///////
#include <iostream>  // include header file

using namespace std;

int main()
{
    cout << "C++ is better than C.\n";   // C++ sta

    return 0;
}
// End of example

//////////////////////// < PROGRAM 2.1 > //////////
```

```
--------------------Configuration: Test - Win32 Debug--------------------
Compiling...
Test.cpp

Test.obj - 0 error(s), 0 warning(s)
```

Fig. C.7 ⇔ *Output window after compiling and linking*

While compiling a C++ source file the Visual C++ application will prompt a message to build a new workspace. Workspace is nothing but an area where we can have a number of source files, their compilation files and linking files saved altogether known as Project. This will be used when we have to create a application with multiple source files.

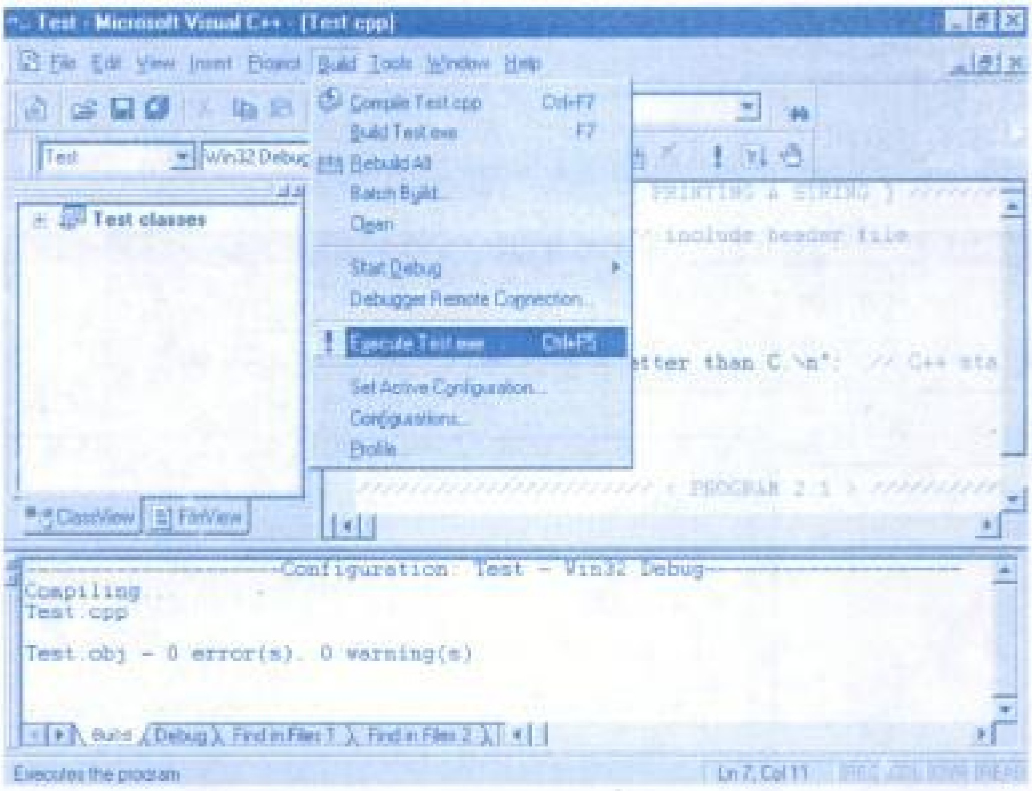
Executable File

The executable file TEST.EXE will be added to the Build menu as shown in the Fig. C.8 after a zero error(s) and zero warning(s) compilation.

The output window indicates that there are no warnings and no errors. The Compile command has successfully generated the executable file TEST.EXE.

C.6 Running the Program

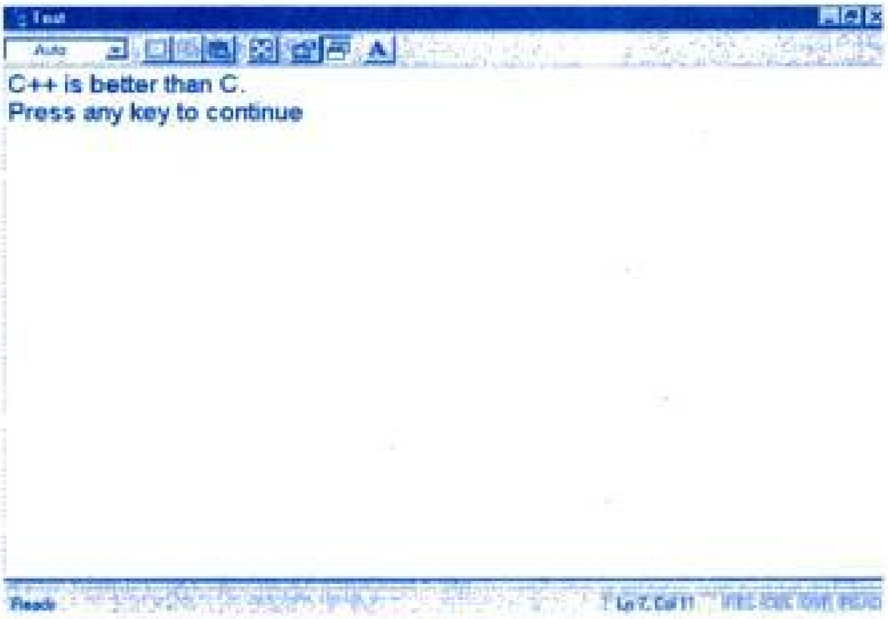
You have reached successfully the final stage of your excitement. Now, to run the program, click the Execute TEST.EXE option in Build menu. The output will be generated in a new windows as shown in Fig. C.9.



```
--------------------Configuration: Test - Win32 Debug--------------------
Compiling...
Test.cpp

Test.obj - 0 error(s), 0 warning(s)
```

Fig. C.8 ⇔ *Build menu after successful compilation*

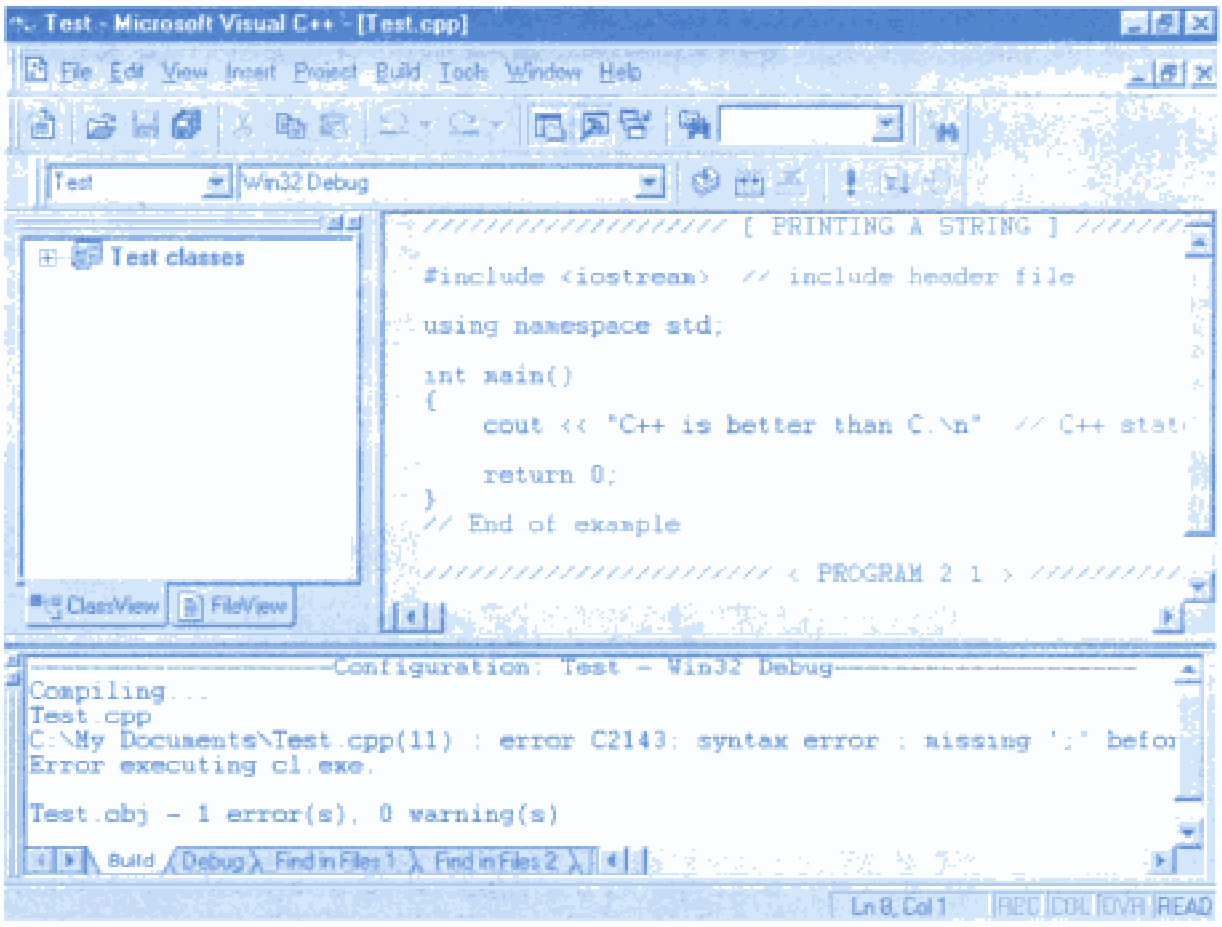


```
C++ is better than C.
Press any key to continue
```

Fig. C.9 ⇔ *Output generated*

C.7 Managing Errors

It is rare that a program runs successfully the first time itself. When the program contains errors, they are displayed in the message window as shown in Fig. C.10.



```
////////////////////// [ PRINTING A STRING ] ////////
#include <iostream>  // include header file
using namespace std;
int main()
{
    cout << "C++ is better than C.\n"  // C++ stat
    return 0;
}
// End of example
//////////////////////// < PROGRAM 2 1 > ///////////
```

```
--------------------Configuration: Test - Win32 Debug--------------------
Compiling...
Test.cpp
C:\My Documents\Test.cpp(11) : error C2143: syntax error : missing ';' befo
Error executing cl.exe.

Test.obj - 1 error(s), 0 warning(s)
```

Fig. C.10 ⇔ *Output window error messages*

You can double-click on a syntax error in the message window to go to the line containing that problem. Fix all the errors, recompile and execute the program.

C.8 Other Features

Windows programmers now have a wider range of tools that can be used for the development of object-oriented systems. Microsoft has provided, among others, the following three tools that would benefit the programmers:

- Foundation Class Library

- Application Wizard
- Class Wizard

The Microsoft Foundation Class (MFC) library contains a set of powerful tools and provides the users with easy-to-use objects. Proper use of MFC library would reduce the length of code and development time of an application.

The AppWizard, short for Application Wizard, helps us to define the fundamental structure of a program and to create initial applications with desired features. However, remember that it only provides a framework and the actual code for a particular application should be written by us.

The ClassWizard, a close associate of the AppWizard, permits us to add classes or customize existing classes. The ClassWizard is normally used after designing the framework using the AppWizard.

It is the power of the Wizards that make the Microsoft Visual C++ so useful and popular. It is therefore important that you are familiar with these tools. You must consult appropriate reference material for complete details.

Appendix D

Glossary of ANSI C++ Keywords

asm	It is to embed the assembly language statements in C++ programs. Its use is implementation dependent.
auto	It is a storage class specifier for the local variables. An auto variable is visible only in the block or function where it is declared. All the local variables are of type auto by default.
bool	It is a data type and is used to hold a Boolean value, **true** or **false.**
break	A **break** statement is used to cause an exit from the loop and switch statements. It is used to provide labels in a switch statement.
catch	**catch** is used to describe the *exception handler* code that catches the exceptions (unusual conditions in the program).
char	It is a fundamental data type and is used to declare character variables and arrays.
class	**class** is used to create user-defined data types. It binds together data and functions that operate on them. Class variables known as *objects* are the building blocks of OOP in C++.
const	It is a data type qualifier. A data type qualified as **const** may not be modified by the program.
const_cast	It is a casting operator used to explicitly override **const** or **volatile** objects.
continue	It causes skipping of statements till the end of a loop in which it appears. It is similar to saying "go to end of loop".
default	It is a **default** label in a switch statement. The control is transferred to this statement when none of the case labels match the expressions in switch.
delete	It is an operator used to remove the objects from memory that were created using new operator.

do — **do** is a control statement that creates a loop of operations. It is used with another keyword *while* in the form:

```
do
{
        statements
}
while(expression);
```

The loop is terminated when the expression becomes zero.

double — It is a floating-point data types specifier. We use this specification to double the number of digits after decimal point of a floating-point value.

dynamic_cast — It is a casting operator used to cast the type of an object at runtime. Its main application is to perform casts on polymorphic objects.

else — **else** is used to specify an alternative path in a two-way branch control of execution. It is used with if statement in the form:

```
if(expression)
        statement-1;
else
        statement-2;
```

The statement-1 is executed if expression is nonzero; otherwise statement-2 is executed.

enum — It is used to create a user-defined integer data type. Example:

```
enum E{e1,e2,...};
```

where e1, e2, ... are enumerators which take integer values. E is a data type and can be used to declare variables of its type.

explicit — It is a specifier to a constructor. A constructor declared as explicit cannot perform implicit conversion.

export — It is used to instantiate non-inline template classes and functions from separate files.

extern — **extern** is a storage class specifier which informs the compiler that the variable so declared is defined in another source file.

false — It is a Boolean type constant. It can be assigned to only a **bool** type variable. The default numeric value of **false** is 0.

loat — It is a fundamental data type and is used to declare a variable to store a single-point precision value.

for — **for** is a control statement and is used to create a loop of iterative operations. It takes the form:

```
for(e1; e2; e3) statement;
```

The *statement* is executed until the expression e2 becomes zero. The expression e1 is evaluated once in the beginning and e3 is evaluated at the end of every iteration.

friend	**friend** declares a function as a **friend** of the class where it is declared. A function can be declared as a friend to more than one class. A **friend** function, although defined like a normal function, can have access to all the members of a class to which it is declared as **friend**.
goto	**goto** is a transfer statement that enables us to skip a group of statements unconditionally. This statement is very rarely used.

if if is a control statement that is used to test an expression and transfer the control to a particular statement depending upon the value of expression. if statement may take one of the following forms:

```
(i) if (expression)
          statement-1;
    statement-2;
(ii) if (expression)
          statement-1;
     else
          statement-2;
```

In form (i), if the expression is nonzero (true), statement-1 is executed and then statement-2 is executed. If the expression is zero (false), statement-1 is skipped. In form (ii), if the expression is nonzero, statement-1 is executed and statement-2 will be skipped; if it is zero, statement-2 is executed and statement-1 is skipped.

inline **inline** is a function specifier which specifies to the compiler that the function definition should be substituted in all places where the function is called.

int It is one of the basic data types and is used to declare a variable that would be assigned integer values.

long **long** is a data type modifier that can be applied to some of the basic data types to increase their size. When used alone as shown below, the variable becomes signed

```
long int.
long m;
```

mutable It is a data type modifier. A data item declared **mutable** may be modified even if it is a member of a **const** object or **const** function.

namespace It is used to define a scope that could hold global identifiers. Example:

```
namespace name
{
        Declaration of identifiers
}
```

new It is an operator used for allocating memory dynamically from free store. We can use new in place of **malloc()** function.

operator	**operator** is used to define an operator function for overloading an operator for use with class objects. Example: `int operator*(vector &v1, vector &v2);`
private	It is a visibility specifier for class members. A member listed under private is not accessible to any function other than the member functions of the class in which it is used.
protected	Like private, protected is also a visibility specifier for class members. It makes a member accessible not only to the members of the class but also to the members of the classes derived from it.
public	This is the third visibility specifier for the class members. A member declared as public in a class is accessible publicly. That is, any function can access a public member.
register	**register** is a storage class specifier for integer data types. It tells the compiler that the object (variable) should be accessible as quickly as possible. Normally, a CPU register is used to store such variables.
reinterpret_cast	It is a casting operator and is used to change one type into a fundamen tally different type.
return	It is used to mark the end of a function execution and to transfer the control back to the calling function. It can also return a value of an expression to the calling function. Example: `return(expression);`
short	Similar to long, it is also a data type modifier applied to integer base types. When used alone with a variable, it means the variable is signed short int.
signed	It is a qualifier used with character and integer base type variables to indicate that the variables are stored with the sign. The high-order bit is used to store the sign bit, 0 meaning positive, 1 meaning negative. A signed char can take values between –127 to +127 whereas an unsigned char can hold values from 0 to 255. The default integer declaration assumes a signed number.
sizeof	**sizeof** is an operator used to obtain the size of a type or an object, in bytes. Example: `int m = sizeof(char);` `int m = sizeof(x);` where x is an object or variable.
static	**static** is a storage class specifier. This can be used on both the local and global variables, but with a different meaning. When it is applied to a local variable, permanent storage is created and it retains its value between function calls in the program. When it is applied to a global variable, the variable becomes internal to the file in which it is declared.
static_cast	It is a casting operator and may be used for any standard conversion of data types.

struct	**struct** is similar to a class and is used to create user-defined data types. It can group together the data items and functions that operate on them. The only difference between a class and struct is that, by default, the struct members are public while the class members become private.
switch	It is a control statement that provides a facility for multiway branching from a particular point. Example: `switch (expression) { case labels }` Depending on the value of expression, the control is transferred to a particular label.
Template	**template** is used to declare generic classes and functions.
this	It is a pointer that points to the current object. This can be used to access the members of the current object with the help of the arrow operator.
Throw	**throw** is used in the exception handling mechanism to "throw" an exception for further action.
true	It is a Boolean type constant. It can be assigned to only a **bool** type variable. The default numeric value of **true** is 1.
try	It is also a keyword used in the exception handling mechanism. It is used to instruct the compiler to try a particular function.
typedef	**typedef** is used to give a new name to an existing data type. It is usually used to write complex declarations easily.
typeid	It is an operator that can be used to obtain the types of unknown objects.
typename	It is used to specify the type of template parameters.
union	It is similar to **struct** in declaration but is used to allocate storage for several data items at the same location.
using	It is a **namespace** scope directive and is used to declare the accessibility of identifiers declared within a **namespace** scope.
unsigned	It is a type modifier used with integer data types to tell the compiler that the variables store non-negative values only. This means that the high-bit is also used to store the value and therefore the size of the number may be twice that of a signed number.
virtual	**virtual** is a qualifier used to declare a member function of a base class as "virtual" in order to perform dynamic binding of the function. It is also used to declare a base class as virtual when it is inherited by a class through multiple paths. This ensures that only one copy of the base class members are inherited.
void	**void** is a data type and is used to indicate the objects of unknown type. Example: `void *ptr;`

is a generic pointer that can be assigned a pointer of any type. It is also used to declare a function that returns nothing. Another use is to indicate that a function does not take any arguments. Example:

```
void print(void);
```

volatile	It is a qualifier used in variable declarations. It indicates that the variable may be modified by factors outside the control of the program.
wchar_t	It is a character data type and is used to declare variables to hold 16-bit wide characters.
while	**while** is a control statement used to execute a set of statements repeatedly depending on the outcome of a test. Example:

```
while (expression)
{
        statements
}
```

The statements are executed until the expression becomes zero.

Appendix E

C++ Operator Precedence

The Table E.1 below lists all the operators supported by ANSI C++ according to their precedence (i.e. order of evaluation). Operators listed first have higher precedence than those listed next. Operators at the same level of precedence (between horizontal lines) evaluate either left to right or right to left according to their associativity.

Table E.1 *C++ Operators*

Operator	*Meaning*	*Associativity*	*Use*
::	global scope	right to left	::name
: :	class, namespace scope	left to right	name : : member
.	direct member	left to right	object.member
–>	indirect member		pointer->member
[ ]	subscript		pointer[expr]
( )	function call		expr(arg)
( )	type construction		type(expr)
++	postfix increment		m++
—	postfix decrement		m––
Sizeof	size of object	right to left	sizeof expr
sizeof	size of type		sizeof (type)
++	prefix increment		++m
––	prefix decrement		––m
typeid	type identification		typeid(expr)
const_cast	specialized cast		const_cast<expr>
dynamic_cast	specialized cast		dynamic_cast<expr>
reinterpret_cast	specialized cast		reinterpret_cast<expr>
static_cast	specialized cast		static_cast<expr>
( )	traditional cast		(type)expr
~	one's complement		~expr

(Contd)

!	logical NOT		! expr
–	unary minus		– expr
+	unary plus		+ expr
&	address of		& value
*	dereference		* expr
new	create object		new type
new [ ]	create array		new type [ ]
delete	destroy object	right to left	delete ptr
delete [ ]	destroy arrary		delete [ ] ptr
.*	member dereference	left to right	object.*ptr_to_member
–>*	indirect member dereference		ptr->*ptr_to_member
*	Multiply	left to right	expr1 * expr2
/	Divide		expr1 / expr2
%	Modulus		expr1 % expr2
+	add	left to right	expr1 + expr2
–	subtract		expr1 – expr2
<<	left shift	left to right	expr1 << expr2
>>	right shift		expr1 >> expr2
<	less than	left to right	expr1 < expr2
<=	less than or equal to		expr1 <= expr2
>	greater than		expr1 > expr2
>=	greater than or equal to		expr1 >= expr2
==	equal	left to right	expr1 == expr2
!=	not equal		expr1 != expr2
&	bitwise AND	left to right	expr1 & expr2
^	bitwise XOR	left to right	expr1 ^ expr2
|	bitwise OR	left to right	expr1 | expr2
&&	logical AND	left to right	expr1 && expr2
| |	logical OR	left to right	expr1 | | expr2
?:	conditional expression	left to right	expr1 ? expr2: expr3
=	assignment	right to left	x = expr
*=	multiply update		x *= expr
/=	divide update		x /= expr
%=	modulus update		x %= expr
+=	add update		x += expr
–=	substract update		x – = expr
<<=	left shift update		x <<= expr
>>=	right shift update		x >>= expr
&=	bitwise AND update		x &= expr
|=	bitwise OR update		x |= expr
^=	bitwise XOR update		x ^= expr
throw	throw exception	right to left	throw expr
,	comma	left to right	expr1, expr2

Appendix F

Points to Remember

1. Computers use the binary number system which uses binary digits called as bits.
2. The basic unit of storage in a computer is a byte represented by eight bits.
3. A computer language is a language used to give instructions to a computer.
4. A compiler translates instructions in programming language to instructions in machine language.
5. Application software is a software that is designed to solve a particular problem or to provide a particular service.
6. Systems software is a software that is designed to support the development and execution of application programs.
7. An operating system is a system software that controls and manages the computing resources such as the memory, the input and output devices, and the CPU.
8. An algorithm is a detailed, step-by-step procedure for solving a problem.
9. The goal of a software design is to produce software that is reliable, understandable, cost effective, adaptable, and reusable.
10. Abstraction is the process of highlighting the essential, inherent aspects of an entity while ignoring irrelevant details.
11. Encapsulation (or information hiding) is the process of separating the external aspects of an object from the internal implementation details which should be hidden from other objects.
12. Modularity is the process of dividing a problem into smaller pieces so that each smaller module can be dealt with individually.
13. Organizing a set of abstractions from most general to least general is known as *inheritance hierarchy*.
14. Object-oriented programming is a paradigm in which a system is modeled as a set of objects that interact with each other.
15. In C++ an abstraction is formed by creating a class. A class encapsulates the attributes and behaviors of an object.
16. The data members of a class represent the attributes of a class.

17. The member functions of a class represent the behaviors of a class.
18. A base class is one from which other, more specialized classes are derived.
19. A derived class is one that inherits properties from a base class.
20. Polymorphism is the capability of something to assume different forms. In an object-oriented language, polymorphism is provided by allowing a message or member function to mean different things depending on the type of object that receives the message.
21. Instantiation is the process of creating an object from a class.
22. We must use the statement #include <iostream> a preprocessor directive that includes the necessary definitions for performing input and output operations.
23. The C++ operator << , called the insertion operator, is used to insert text into an output stream.
24. The C++ operator >>, called the extraction operator, is used to insert text into an input stream.
25. All C++ programs begin executing from the **main**. Function **main** returns an integer value that indicates whether the program executed successfully or not. A value of 0 indicates successful execution, while the value 1 indicates that a problem or error occurred during the execution of the program.
26. A value is returned from a function using the **return** statement. The statement

```
return 0;
```

returns the value 0.
27. A C++ style comment begins with // and continues to the end of the line.
28. A C++ identifiers consists of a sequence of letters (upper and lowercase), digits, and underscores. A valid name cannot begin with a digit character.
29. C++ identifiers are case sensitive. For example, **Name** and **name** refer to two different identifiers.
30. A variable must be defined before it can be used. Smart programmers give a variable an initial value when it is defined.
31. The automatic conversion specifies that operands of type **char** or **short** are converted to type **int** before proceeding with operation.
32. For an arithmetic operation involving two integral operands, the automatic conversion specifies that when the operands have different types, the one that is type **int** is converted to **long** and a **long** operation is performed to produce a **long** result.
33. For an arithmetic operation involving two floating-point operands, the automatic conversion specifies that when the operands are of different types, the operand with lesser precision is converted to the type of the operand with greater precision.
34. A mixed-mode arithmetic expression involves integral and floating-point operands. The integral operand is converted to the type of the floating-point operand, and the appropriate floating-point operation is performed.
35. The precedence rules of C++ define the order in which operators are applied to operands. For the arithmetic operators, the precedence from highest to lowest is unary plus and minus; multiplication, division, and modules; and addition and subtraction.
36. It is a good programming practice to initialize a variable or an object when it is declared.

Hidden page

Hidden page

Hidden page

101. The location of a variable can be obtained using the address operator &.
102. The literal 0 can be assigned to any pointer type object. In this context, the literal 0 is known as the null address.
103. The value of the object at a given location can be obtained using the indirection operator * on the location.
104. The indirection operator produces an lvalue.
105. The null address is not a location which can be dereferenced.
106. The member selector operator -> allows a particular member of object to be dereferenced.
107. Pointer operators may be compared using the equality and relational operators.
108. The increment and decrement operators may be applied to pointer objects.
109. Pointers can be passed as reference parameters by using the indirection operator.
110. An array name is viewed by C++ as constant pointer. This fact gives us flexibility in which notation to use when accessing and modifying the values in a list.
111. Command-line parameters are communicated to programs using pointers.
112. We can define variables that are pointers to functions. Such variables are typically used as function parameters. This type of parameter enables the function that uses it to have greater flexibility in accomplishing its task.
113. Increment and decrement of pointers follow the pointer arithmetic rules. If **ptr** points to the first element of an array, then **ptr+1** points to the second element.
114. The name of an array of type **char** contains the address of the first character of the string.
115. When reading a string into a program, always use the address of the previously allocated memory. This address can be in the form of an array name or a pointer that has been initialized using **new**.
116. Structure members are **public** by default while the class members are **private** by default.
117. When accessing the class members, use the dot operator if the class identifier is the name of the class and use the arrow operator if the identifier is the pointer to the class.
118. Use **delete** only to delete the memory allocated by **new**.
119. It is a good practice to declare the size of an array as a constant using the qualifier **const**.
120. C++ supports two types of parameters, namely, value parameters and reference parameters.
121. When a parameter is passed by value, a copy of the variable is passed to the called function. Any modifications made to the parameter by the called function change the copy, not the original variable.
122. When a reference parameter is used, instead of passing a copy of the variable, a reference to the original variable is passed. Any modifications made to the parameter by the called function change the original variable.
123. When an **iostream** object is passed to a function, either an extraction or an insertion operation implicitly modifies the stream. Thus, stream objects should be passed a reference.
124. A reason to use a reference parameter is for efficiency. When a class object is passed by value, a copy of the object is passed. If the object is large, making a copy of it can

be expensive in terms of execution time and memory space. Thus objects that are large, or objects whose size is not known are often passed by reference. We can ensure that the objects are not modified by using the **const** modifier.

125. A const modifier applied to a parameter declaration indicates that the function may not change the object. If the function attempts to modify the object, the compiler will report a compilation error.
126. A reference variable must be initialized when it is declared.
127. When you are returning an address from a function, never return the address of local variable though, syntactically, this is acceptable.
128. If a function call argument does not match the type of a corresponding reference parameter, C++ creates an anonymous variable of the correct type, assigns the value of the argument to it and causes the reference parameter to refer the variable.
129. A function that returns a reference is actually an alias for the "referred-to" variable.
130. We can assign a value to a C++ function if the function returns a reference to a variable. The value is assigned to the referred-to variable.
131. C++'s default parameter mechanism provides the ability to define a function so that a parameter gets a default value if a call to the function does not give a value for that parameter.
132. Function overloading occurs when two or more function have the same name.
133. The compiler resolves overloaded function calls by calling the function whose parameters list best matches that of the call.
134. Casting expressions provide a facility to explicitly convert one type to another.
135. A cast expression is useful when the programmer wants to force the compiler to perform a particular type of operation such as floating-point division rather than integer division.
136. A cast expression is useful for converting the values that library function return to the appropriate type. This makes it clear to other programmers that the conversion was intended.
137. An *inline function* must be defined before it is called.
138. An *inline function* reduces the function call overhead. Small functions are best declared inline within a class.
139. In a multiple-file program, you can define an external variable in one and only one file. All the other files using that variable have to declare it with the keyword **extern**.
140. An abstract data type (ADT) is well-defined and complete data abstraction that uses the principle of information-hiding.
141. An ADT allows the creation and manipulation of objects in a natural manner.
142. If a function or operator can be defined such that it is not a member of the class, then do not make it a member. This practice makes a nonmember function or operator generally independent of changes to the class's implementation.
143. In C++, an abstract data type is implemented using classes, functions, and operators.
144. Constructors initialize objects of the class type. It is standard practice to endure that every object has all of its data members appropriately initialized.
145. A default constructor is a constructor that requires no parameters.

146. A copy constructor initializes a new object to be a duplicate of a previously defined source object. If a class does not define a copy constructor, the compiler automatically supplies a version.
147. A member assignment operator copies a source object to the invoking target object in an assignment statement. If a class does not define a member assignment operator, the compiler automatically supplies a version.
148. When we call a member function, it uses the data members of the object used to invoke the member function.
149. A class constructor, if defined, is called whenever a program creates an object of that class.
150. When we create constructors for a class, we must provide a default constructor to create uninitialized objects.
151. When we assign one object to another of the same class, C++ copies the contents of each data member of the source object to the corresponding member of the target object.
152. A member function operates upon the object used to invoke it, while a friend function operates upon the objects passed to it as arguments.
153. The qualifier **const** appended to function prototype indicates that the function does not modify any of the data members. A **const** member function can be used by **const** objects of the class.
154. The client interface to a class object occurs in the **public** section of the class definition.
155. Any member defined in any section — whether **public, protected,** or **private** — is accessible to all of the other members of its class.
156. Members of a **protected** section are intended to be used by a class derived from the class.
157. Data members are normally declared in a **private**. By restricting outside access to the data members in a class, it is easier to ensure the integrity and consistency of their values.
158. Members of **private** section of a class are intended to be used only by the members of that class.
159. An & in the return type for a function or operator indicates that a reference return is being performed. In a reference return, a reference to the actual object in the return expression rather than a copy is returned. The scope of the returned object should not be local to the invoked function or operator.
160. When creating a **friend** function, use the keyword friend in the prototype in the class definition, but do not use this keyword in the actual function definition. Friend functions are defined outside the class definition.
161. Friend functions have access to the private and protected members of a class.
162. An operator can be overloaded many times using distinct signatures.
163. If we want to overload a binary operator with two different types of operands with non class as the first operand, we must use a friend function to define the operator overloading.
164. Do not use implicit type conversions unless it is necessary. If they are used arbitrarily, it can cause problems for future users of the class.

165. Whenever we use **new** in a constructor to allocate memory, we should use **delete** in the corresponding destructor to free that memory.
166. The relationship "is_ a" indicates inheritance. For example, a car is a kind of vehicle.
167. The relationship "has_a" indicates containment. For example, a car has an engine. Aggregate objects are constructed using containment.
168. Both inheritance and containment facilitate software reuse.
169. A new class that is created from an existing class using the principle of inheritance is called a *derived class or subclass*. The parent class is called the *base class or superclass.*
170. When an object that is a instance of derived class is instantiated, the constructor for the base class is invoked before the body of the constructor for the derived class is invoked.
171. A class intended to be a base class usually should use **protected** instead of **private** members.
172. When a derived class object is being created, first its base classes constructors are called before its own constructor. The destructors are called in the reverse order.
173. A constructor of a derived class must pass the arguments required by its base class constructor.
174. A derived class uses the member functions of the base class unless the derived class provides a replacement function with the same name.
175. A derived class object is converted to a base class object when used as an argument to a base class member function.
176. Derived class constructors are responsible for initializing any data members added to those inherited from the base class. The base class constructors are responsible for initializing the inherited data members.
177. When passing an object as an argument to the function, we usually use a reference or a pointer argument to enable function calls within the function to use virtual member function.
178. Declare the destructor of a base class as a virtual function.
179. Destructors are called in reverse order from the constructor calls. Thus, the destructor for a derived class is called before the destructor of the base or superclass.
180. With **public** inheritance, the **public** members of the base class are public members of the derived class. The **private** members of the base class are not inherited and, therefore, not accessible in the derived class.
181. With **protected** inheritance, **public** and **protected** members of the base class become **protected** members of the derived class. The **private** members of the base class are not inherited.
182. With multiple inheritance, a derived class inherits the attributes and behaviors of all parent classes.
183. With **private** inheritance, **public** and **protected** members of the base class become **private** members of the derived class. Private members are not inherited.
184. If a derived class has a base class as a multiple ancestor (through multiple inheritance), then declare the base class as **virtual** in the derived class definition. This would ensure the inheritance of just one object of the base class.
185. A pointer to a base class can be used to access a member of the derived class, as long as that class member is inherited from the base.

Hidden page

Hidden page

234. The member function **eof** of **ios** determines if the end of the file indicator has been set. End-of-file is set after an attempted read fails.
235. To use C++ **strings**, we must include the header file **<string>** of **C++** standard library.
236. C++ strings are not null terminated.
237. Using **STL** containers can save considerable time and effort, and result in higher quality programs.
238. To use containers, we must include appropriate header files.
239. **STL** includes a large number of algorithms to perform certain standard operations on containers.
240. **STL** algorithms use iterators to perform manipulation operations on containers.
241. We may use **const**-cast operator to remove the constantness of objects.
242. We may use **mutable** specifier to the members of **const** member functions or **const** objects to make them modifiable.
243. We must restrict the use of runtime type information functions only with polymorphic types.
244. When we suspect any side-effects in the constructors, we must use **explicit** constructors.
245. We must provide parentheses to all arguments in macro functions.

Appendix G

Glossary of Important C++ and OOP Terms

#define	A C++ preprocessor directive that defines a substitute text for a name.
#include	A preprocessor directive that causes the named file to be inserted in place of the #include.
Abstract Class	A class that serves only as a base class from which classes are derived. No objects of an abstract base class are created. A base class that contains pure virtual functions is an abstract base class.
Abstract Data Type (ADT)	An abstraction that describes a set of objects in terms of an encapsulated or hidden data and operations on that data.
Abstraction	The act of representing the essential features of something without including much detail.
Access Operations	Operations which access the state of a variable or object but do not modify it.
Address	A value that identifies a storage location in memory.
Alias	Two or more variables that refer to the same data are said to be aliases of one another.
Anonymous Union	An unnamed union in C++. The members can be used like ordinary variables.
ANSI C	Any version of C that conforms to the specifications of the American National Standards Institute Committee X3J.
ANSI C++	Any version of C++ that conforms to the specifications of the American National Standards Institute. At the time of writing this, the standards exist only in draft form and a lot of details are still to be worked out.
Array	A collection of data elements arranged to be indexed in one or more dimensions. In C++, arrays are stored in contiguous memory.
ASCII	American Standard Code for Information Interchange. A code to represent characters.

Assignment Statement	An operation that stores a value in a variable.
Attribute	A property of an object. It cannot exist independently of the object. Attributes may take other objects as values.
Automatic Variable	*See* temporary variable.
Base Class	A class from which other classes are derived. A derived class can inherit members from a base class.
Bit	Binary digit; either of the digits 0 or 1.
Bit Field	A group of contiguous bits taken together as a unit. This C++ language feature allows the access of individual bits.
Bit Flip	The inversion of all bits in an operand. See also complement.
Bitmapped Graphics	Computer graphics where each pixel in the graphic output device is controlled by a single bit or a group of bits.
Bitwise Operator	An operator that performs Boolean operations on two operands, treating each bit in an operand as individual bits and performing the operation bit by bit on corresponding bits.
Block	A section of code enclosed in curly braces.
Borland C++	A version of the C++ language for personal computers developed by Borland. This is the high-end version of Borland's Turbo-C++ product.
Breakpoint	A location in a program where normal execution is suspended and control is turned over to the debugger.
Byte	A group of eight bits.
C	A general-purpose computer programming language developed in 1974 at Bell Laboratories by Dennis Ritchie. C is considered to be medium-to high level language.
C++	An object-oriented language developed by Bjarne Stroutstrup as a successor of C.
Call by Reference	A function call mechanism that passes arguments to a function by passing the addresses of the arguments.
Call by Value	A function call mechanism that passes arguments to a function by passing a copy of the value of the arguments.
Cast	To convert a variable from one type to another type by explicitly.
Class	A group of objects that share common properties and relationships. In C++, a class is a new data type that contains member variables and member 0 functions that operate on the variables. A Class is defined with the keyword **class**.
Class Hierarchy	Class hierarchy consists of a base class and derived classes. When a derived class has a single base class, it is known as single inheritance.

	When a derived class has more than one base class (multiple inheritance), it is known as **class network**.
Class Network	A collection of classes, some of which are derived from others. A class network is a class hierarchy generalized to allow for multiple inheritance. It is sometimes known as forest model of classes.
Class Object	A variable whose type is a class. An instance of a class.
Classification structure	A tree or network structure based on the semantic primitives of inclusion and membership which indicates that inheritance may implement specialization or generalization. Objects may participate in more than one such structure, giving rise to multiple inheritance.
Class-oriented	Object-based systems in which every instance belongs to a class, but classes may not have super classes.
Client	An object that uses the services of another object called server. That is, clients can send messages to servers.
Coding	The act of writing a program in a computer language.
Comment	Text included in a computer program for the sole purpose of providing information about the program. Comments are a programmer's notes to himself and future programmers. The text is ignored by the compiler.
Comment Block	A group of related comments that convey general information about a program or a section of program.
Compiliation	The translation of source code into machine code.
Compiler	A system program that does compilation.
Complement	An arithmetic or logical operation. A logical complement is the same as an invert or NOT operation.
Composition Structure	A tree structure based on the semantic primitive part of' which indicates that certain objects may be assembled from the collection of other objects. Objects may participate in more than one such structure.
Conditional Compilation	The ability to selectively compile parts of a program based on the truth of conditions tested in conditional directives that surround the code.
Constructor	A special member function for automatically creating an instance of a class. This function has the same name as the class.
Container Class	A class that contains objects of other classes.
Control Statement	A statement that determines which statement is to be executed next based on a conditional test.
Control Variables	A variable that is systematically changed during the execution of the loop. When the variable reaches a predetermined value, the loop is terminated.

Copy Constructuor	The constructor that creates a new class object from an existing object of the same class.
Curly Braces	One of the characters { or }. They are used in C++ to delimit groups of elements to treat them as a unit.
Data Flow Diagram (DFD)	A diagram that depicts the flow of data through a system and the processes that manipulate the data.
Data Hiding	A property whereby the internal data structure of an object is hidden from the rest of the program. The data can be accessed only by the functions declared within the class (of that object).
Data Member	A variable that is declared in a class declaration.
Debugging	The process of finding and removing errors from a program.
Decision Statement	A statement that tests a condition created by a program and changes the flow of the program based on that decision.
Declaration	A specification of the type and name of a variable to be used in a program.
Default Argument	An argument value that is specified in a function declaration and is used if the corresponding actual argument is omitted when the function is called.
De-referencing Operator	The operator that indicates access to the value pointed to by a pointer variable or an addressing expression. *See* also indirection operator.
Derived Class	A class that inherits some or all of its members from another class, called **base class**.
Destructor	A function that is called to deallocate memory of the objects of a class.
Directive	A command to the preprocessor (as opposed to a statement to produce machine code).
Dynamic Binding	The addresses of the functions are determined at run time rather than compile time. This is also known as late binding.
Dynamic Memory Allocation	The means by which data objects can be created as they are needed during the program execution. Such data objects remain in existence until they are explicitly destroyed. In C++, dynamic memory allocation is accomplished with the operators **new** (for creating data objects) and **delete** (for destroying them).
Early Binding	*See* static binding.
Encapsulation	The mechanism by which the data and functions (manipulating this data) are bound together within an object definition.
Enumerated Data Type	A data type consisting of a named set of values. The C++ compiler assigns an integer to each member of the set.
Error State	For a stream, flags that determine whether an error has occurred and, if so, give some indication of its severity.

Hidden page

Heterogeneous List	A list of class objects, which can belong to more than one class. Processing heterogeneous lists is an important application of polymorphism.
Homogeneous List	A list of class objects all of which belong to the same class.
I/O Manipulators	Functions that when "output" or "input" cause no I/O, but set various conversion flags or parameters.
Implementation	The source code that embodies the realization of the design.
Include File	A file that is merged with source code by invocation of the preprocessor directive #include. Also called a header file.
Index	A value, variable or expression that selects a particular element of an array.
Indirect Operator	*See* de-referencing operator.
Indirection Operator	The operator *, which is used to access a value referred to by a pointer.
Information Hiding	The principle which states that the state and implementation of an object or module should be private to that object or module and only accessible via its public interface. See encapsulation.
Inheritance	A relationship between classes such that the state and implementation of an object or module should be private to that object or module and only accessible via its public interface. *See* encapsulation.
Inheritance Path	A series of classes that provide a path along which inheritance can take. For example, if class B is derived from A, class C is derived from class B, and class D is derived from class C, then class D inherits from class A via the inheritance path ABCD.
Initialization List	In the definition of a constructor, the function heading can be followed by a colon and a list of calls to other constructors. This initialization list can contain calls to (1) constructors for base classes and (2) constructors for class members that are themselves class objects.
Inline Function	A function definition such that each call to the function is, in effect, replaced by the statements that define the function.
Insertion Operator	The operator <<, which is used to send output data to the screen.
Instance	An instance of a class is an object whose type is the class in question.
Instance Variable	A data member that is not designated as static. Each instance of a class contains a corresponding data object for each nonstatic data member of the class. Because the data objects are associated with each instance of the class, rather than with the class itself, we refer to them as **instance** variables.
Instantiation	The creation of a data item representing a variable or a class (giving a value to something).

Hidden page

Hidden page

Hidden page

This	This is a pointer to the current object. It is passed implicitly to an overloaded operator function.
Translation	Creation of a new program in an alternate language logically equivalent to an existing program in a source language.
Truncation	An operation on a real number whereby any fractional part is discarded.
Turbo C++	A version of the C++ language for personal computers developed by Borland.
Type Conversion	A conversion of a value from one type to another.
Typecast	*See* cast.
Typedef Name	A name given to a type via a type-name definition introduced by the key-word **typedef.**
Union	A data type that allows different data types to be assigned to the same storage location.
Value	A quantity assigned to a constant.
Variable	A name that refers to a value. The data represented by the variable name can, at different times during the execution of a program, assume different values.
Variable Name	The symbolic name given to a section of memory used to store a variable.
Virtual Base Class	A base class that has been qualified as virtual in the inheritance definition. In multiple inheritance, a derived class can inherit the members of a base class via two or more inheritance paths. If the base class is not virtual, the derived class will inherit more than one copy of the members of the base class. For a virtual base class, however, only one copy of its members will be inherited regardless of the number of inheritance paths between the base class and the derived class.
Virtual Function	A function qualified by the **virtual** keyword. When a virtual function is called via a pointer, the class of the object pointed to determines which function definition will be used. Virtual functions implement polymorphism, whereby objects belonging to different classes can respond to the same message in different ways.
Visibility	The ability of one object to be a server to others.
Void	A data type in C++. When used as a parameter in a function call, it indicates there is no return value. void+ indicates that a generic pointer value is returned. When used in casts, it indicates that a given value is to be discarded.
Windows	A graphical partition of screen for user interface.

Appendix H

C++ Proficiency Test

Part A

True / False Questions

State whether the following statements are true or false

1. A C++ program is identical to a C program with minor changes in coding
2. Bundling functions and data together is known as data hiding.
3. In C++, a function contained within a class is called a member function.
4. Object modeling depicts the real-world entities more closely than do functions.
5. In using object-oriented languages like C++, we can define our own data types.
6. When a C++ program is executed, the function that appears first in the program is executed first.
7. In a 32-bit system, the data types **float** and **long** occupy the same number of bytes.
8. In an assignment statement such as int x = expression; the value of x is always equal to the value of the expression on the right.
9. In C++, declarations can appear almost anywhere in the body of a function.
10. C++ does not permit mixing of variables of different data types in an arithmetic expression.
11. The value of the expression 13%4 is 3.
12. Assuming the value of variable x as 10, the output of the statement `cout << x--;` will be 10.
13. The expression **for(; ;)** is the same as a **while** loop with a test expression of **true**.
14. In C++, arithmetic operators have a lower precedence than relational operators.
15. In C++, only **int** type variables can be used as loop control variables in a **for** loop.
16. A **do** loop is executed at least once.

17. The **&&** and `||` operators compare two boolean values.
18. The control variable of a **for** loop can be decremented inside the **for** statement.
19. The **break** statement is used to exit from all the nested loops.
20. The **default** case is required in the **switch** selection structure.
21. The **continue** statement inside a **for** loop transfers the control to the top of the loop.
22. The **goto** statement cannot be used to transfer the control out of a nested loop.
23. A conditional expression such as `(x < y) ? x : y` can be used anywhere a value can be.
24. A structure and a class use similar syntax.
25. Memory space for a structure member is created when the structure is declared.
26. If **item1** and **item2** are variables of type structure **Item**, then the assignment operation `item1 = item2;` is legal.
27. When calling a function, if the arguments are passed by reference, the function works with the actual variables in the calling program.
28. A structure variable cannot be passed as an argument to a function.
29. A C++ function can return multiple values to the calling function.
30. A function call of a function that returns a value can be used in an expression like any other variable.
31. We need not specify any return type for a function that does not return anything.
32. A set of functions with the same return type are called overloaded functions.
33. Only when an argument has been initialized to zero value, it is called the default argument.
34. A variable declared above all the functions in a program can be accessed only by the **main( )** function.
35. A static automatic variable retains its value even after exiting the function where it is defined.
36. We can use a function call on the left side of the equal sign when the function returns a value by reference.
37. Returning a reference to an automatic variable in a called function is a logic error.
38. Reference variables should be initialized when they are declared.
39. Using **inline** functions may reduce execution time, but may increase program size.
40. A C++ array can store values of different data types.
41. Referring to an element outside the array bounds is a syntax error.
42. When an array name is passed to a function, the function access a copy of the array passed by the program.
43. The extraction operator >> stops reading a string when a space is encountered.
44. Objects of the **string** class can be copied with the assignment operator.
45. Strings created as objects of the **string** class are zero-terminated.
46. Pointers of different types may not be assigned to one another without a cast operation.
47. Not initializing a pointer when it is declared is a syntax error.
48. Data members in a class must be declared **private**.
49. Data members of a class cannot be initialized in the class definition.

50. Members declared as **private** in a class are accessible to all the member functions of that class.
51. In a class, we cannot have more than one constructor with the same name.
52. A member function declared **const** cannot modify any of its class's member data.
53. In a class, members are private by default.
54. In a structure, members are public by default.
55. A member variable defined as **static** is visible to all classes in the program.
56. An object declared as **const** can be used only with the member functions that are also declared as **const**.
57. A member function can be declared **static**, if it does not access any non-static class members.
58. A non member function may have access to the **private** data of a class if it is declared as a **friend** of that class.
59. The precedence of an operator can be changed by overloading it.
60. Using the keyword **operator**, we can create new operators in C++.
61. We can convert a user-defined class to a basic type by using a one-argument constructor.
62. We can always treat a base-class object as a derived-class object.
63. A derived class cannot directly access the **private** members of its base class.
64. In inheritance, the base-class constructors are called in the order in which inheritance is specified in the derived class definition.
65. Inheritance is used to improve data hiding and encapsulation.
66. We can convert a base-class pointer to a derived class pointer using a cast.
67. When deriving a class from a base class with **protected** inheritance, **public** members of the base class became **protected** members of the derived class.
68. When deriving a class from a base class with **public** inheritance, **protected** members of the base class become **public** members of the derived class.
69. A **protected** member of a base class cannot be accessed from a member function of the derived class.
70. In case constructors are not specified in a derived class, the derived class will use the constructors of the base class for constructing its objects.
71. The scope-resolution operator tells us what base class a class is derived from.
72. A derived class is often called a subclass because it represents a subset of its base class.
73. It is permitted to make an object of one class a member of another class.
74. Virtual functions permit us to use the same function call to execute member functions of different classes.
75. A pointer to a base class can point to an object of a derived class of that base class.
76. An **abstract** class is never used as a base class.
77. A pure virtual function in a class will make the class abstract.
78. A derived class can never be made an **abstract** class.
79. A **static** function can be invoked using its class name and function name.
80. The input and output stream features are provided as a part of C++ language.
81. A file pointer always contains the address of the file.

82. Templates create different versions of a function at runtime.
83. Template classes can work with different data types.
84. A template function can be overloaded by another template function with the same function name.
85. A function template can have more than one template argument.
86. Class templates can have only class-type as parameters.
87. A program cannot continue to execute after an exception has occurred.
88. An exception is always caused by a syntax error.
89. After an exception is processed, control will return to the first statement after the **throw**.
90. An exception should be thrown only within a **try** block.
91. If no exceptions are thrown in a **try** block, the **catch** blocks for that **try** block are skipped and the control goes to the first statement after the last **catch** block.
92. The statement **throw**; rethrows an exception.
93. Two **catch** handlers cannot have the same type.
94. Exceptions are thrown from a **throw** statement to a **catch** block.
95. STL algorithms can work successfully with C-like arrays.
96. Algorithms can be added easily to the STL, without modifying the container classes.
97. A **map** can store more than one element with the same key value.
98. A **vector** can store different types of objects.
99. In an associative container, the keys are stored in sorted order.
100. In a **deque**, data can be quickly inserted or deleted at either end.
101. Two functions cannot have the same name in ANSI C++.
102. The modulus operator(%) can be used only with integer operands.
103. Declarations can appear anywhere in the body of a C++ function.
104. All the bitwise operators have the same level of precedence in Java.
105. If $a = 10$ and $b = 15$, then the statement $x = (a > b)\ ?\ a : b$; assigns the value 15 to x.
106. In evaluating a logical expression of type `boolean expression - 1 && boolean expression - 2` both the boolean expressions are not always evaluated.
107. In evaluating the expression ($x == y$ && $a < b$) the boolean expression $x == y$ is evaluated first and then $a < b$ is evaluated.
108. The **default** case is required in the **switch** selection structure.
109. The **break** statement is required in the default case of a **switch** selection structure.
110. The expression ($x == y$ && $a < b$) is true if either $x == y$ is true or $a < b$ is true.
111. A variable declared inside the **for** loop control cannot be referenced outside the loop.
112. Objects are passed to a function by use of call-by-reference only.
113. We can overload functions with differences only in their return type.
114. It is an error to have a function with the same signature in both the super class and its subclass.
115. Derived classes of an abstract class that do not provide an implementation of a pure virtual function are also abstract.
116. Members of a class specified as **private** are accessible only to the functions of the class.

Hidden page

Hidden page

46. ________________ is a way to add features to existing classes without rewriting them.
47. When the class B is inherited from the class A, class A is called the ________________ class and class B is called the ________________ class.
48. The process of inheriting features from many basic classes is known as ________________.
49. The members declared as ________________ or ________________ in the base class may be accessed from a member function of the derived class.
50. In protected derivation, public members of the base class become ________________ members of the derived class.
51. In a multipath inheritance, the duplication of inherited members from the grandparent class can be avoided by declaring the grandparent class as ________________ while declaring the intermediate base classes.
52. A class that is designed only to act as a base class but not used to create objects is known as ________________ class.
53. Inheritance represents ________________ relationship between classes and composition represents ________________ relationship between classes.
54. The ________________ operator is used to specify a particular class.
55. A function call resolved at run time is referred to as ________________ binding.
56. When we use the same function name in both the base and derived classes dynamic binding is achieved by declaring the base class function as ________________.
57. A ________________ function causes its class to be abstract.
58. A virtual function can be made pure virtual function by placing ________________ at the end of its prototype in the class definition.
59. The only integer that can be assigned to a pointer is ________________.
60. A pointer is a variable for storing ________________.
61. The content of an int type pointer increases by ________________ bytes whenever the increment operator is applied to it.
62. A pointer to ________________ can hold pointers to any data type.
63. While passing arguments to a function, passing them by pointers allow the function to ________________ the arguments in the calling function.
64. The base class for most of the input and output stream classes is the ________________ class.
65. Output operations are supported by the ________________ class.
66. The class ________________ declares input functions such as get() and read().
67. When using manipulator functions to alter the output format parameters of streams, we must include the header file ________________.
68. The default precision for printing floating point numbers is ________________ digits.
69. The flag ________________ causes the display of trailing zeros.
70. To write data that contains variables of type to an object of type of stream, we should use ________________ function.
71. The function ________________ writes a single character to the associated stream.

72. To place the input pointer in a specified location in the file, we must use the ____________________ function.
73. Opening a file in ios::out mode also opens it in the ____________________ mode by default.
74. The read() and write() functions handle data in ____________________ form.
75. We must open the file using ____________________ option for performing both input and output operations.
76. Command-line arguments are accessed through arguments to ____________________.
77. A ____________________ provides a convenient way to create a family of classes and functions.
78. A function template definition begin with the keyword ____________________.
79. A call instantiated from a class template is called a ____________________.
80. All functions instantiated from a function template have the same name; therefore, the compiler applies the concept of ____________________ resolution to invoke the required function.
81. A template argument is preceded by the keyword ____________________.
82. A template function works with ____________________ data types.
83. An exception is typically caused by ____________________ error.
84. Exception are thrown from a ____________________ statement to a ____________________ block.
85. The code that is likely to produce an exception is enclosed in a ____________________ block.
86. The catch handler ____________________ will catch all types of exceptions.
87. By default, if no handler is found for an exception, the program ____________________.
88. The container deque is a ____________________ type container.
89. The three STL container adapters are stack, queue, and ____________________.
90. The STL algorithms operate on container elements indirectly using ____________________.
91. A ____________________ is an appropriate container if we are given an element's key value and we want to quickly access the corresponding value.
92. In a ____________________ container, the data can be quickly inserted or deleted at either end.
93. In ____________________ containers, keys are stored in sorted order.
94. For using function objects, we must include the header file ____________________.
95. For using the algorithm accumulate(), we must include the header file ______________.
96. The ____________________ operator is used to change the constantness of objects.
97. The operator ____________________ returns a reference to a type-info object.
98. Non standard casts between unrelated types may be achieved by using the operator ____________________.
99. The operator ____________________ qualifies a member with its namespace.
100. The use of specifier ____________________ to a data item permits us to modify it even when it is a member of a const object.

Hidden page

Hidden page

Hidden page

Hidden page

Hidden page

E. To hide the details of base classes

42. Consider the following class definition.

```
class Person
{
};
class Student : protected Person
{
};
```

What happens when we try to compile this class?

A. Will not compile because class body of person is not defined
B. Will not compile because the class body of Student is not defined
C. Will not compile because class Person is not public inherited
D. Will compile successfully.

43. Consider the following class definitions:

```
class Maths
{
      Student student1;
};
class Student
{
      String name;
};
```

This code represents:

A. an 'is a' relationship
B. a 'has a' relationship
C. both
D. neither

44. Which of the following are overloading the function

```
      int sum(int x, int y)  {  }
```

A. int sum(int x, int y, int z) { }
B. float sum(int x, int y) { }
C. int sum(float x, float y) { }
D. int sum(int a, int b) { }
E. float sum(int x, int y, float z) { }

45. What is the error in the following code?

```
class Test
{
      virtual void display( );
}
```

A. No error
B. Function **display( )** should be declared as **static**
C. Function display() should be defined
D. **Test** class should contain data members

46. Which of the following declarations are illegal?
A. void *ptr;
B. char *str1 = "xyz";
C. char str2 = "abc";
D. const *int p1;
E. int * const p2;

47. The function **show()** is a member of the class **A** and **abj** is a object of A and **ptr** is a pointer to A. Which of the following are valid access statements?
A. abj.show();
B. abj→show();
C. ptr→show();
D. ptr.show();
E. ptr*show();
F. (*ptr).show();

48. We can make a class abstract by
A. Declaring it abstract using the static keyword
B. Declaring it abstract using the virtual keyword
C. Making at least one member function as virtual function
D. Making at least one member function as pure virtual function
E. Making all member functions **const**.

49. Consider the following code:

```
class A
{ public : virtual void show() = 0; };

class B : public A
{ public : void display()
     { cout << "B"; }  };

class C : public A
{  public : void show()
      { cout << "C"; }  };
```

Which of the following statements are illegal?
A. C c1;
B. A a1;
C. B b1;
D. A * arr[2];
E. arr[0] = &c1;

Hidden page

57. Which of the following keywords are used to control access to a class member?
    A. default
    B. break
    C. protected
    D. goto
    E. public
58. Which of the following keywords were added by ANSI C++?
    A. asm
    B. explicit
    C. enum
    D. extern
    E. typename
    F. using
59. Which of the following statements are valid array declaration?
    A. int number(5);
    B. float average[5];
    C. double[5] marks;
    D. counter int[5];
    E. int x[5], y[10];
60. What will be the content of array variable table after executing the following code

```
for(int i=0; i<3; i++)
     for(int j=0, j<3; j++)
            if(j == i) table[i][j] = 1;
            else table[i][j] = 0;
```

```
A. 0 0 0        B. 1 0 0        C. 0 0 1        D. 1 0 0
   0 0 0           1 1 0           0 1 0           0 1 0
   0 0 0           1 1 1           1 0 0           0 0 1
```

61. Which of the following methods belong the **string** class?
    A. length( )
    B. compareTo( )
    C. equals( )
    D. substring( )
    E. All of them
    F. None of them
62. Given the code

```
string s1 = "yes";
string s2 = "yes";
string s3 = string s3(s1);
```

Which of the following would equate to **true**?
    A. s1 == s2

B. s1 = s2
C. s3 == s1
D. s1.equals(s2)
E. s3.equals(s1)

63. Suppose that s1 and s2 are two strings. Which of the statements or expressions are correct?
A. string s3 = s1 + s2;
B. string s3 = s1 – s2;
C. s1 <= s2
D. s1.compareTo(s2);
E. int m = s1.length( );

64. Given the code

```
string s("abc");
```

Which of the following calls are valid?
A. s.trim( )
B. s.replace('a', 'A')
C. s.substring(3)
D. s.toUpperCase( )

65. Given the declarations

```
bool b;
int x1 = 100, x2 = 200, x3 = 300;
```

Which of the following statements are evaluated to *true*?
A. b = x1 * 2 == x2;
B. b = x1 + x2 != 3 * x1;
C. b = (x3 - 2*x2 < 0) || ((x3 = 400) < 2*x2);
D. b = (x3 - 2*x2 > 0) || ((x3 = 400) < 2*x2);

66. In which of the following code fragments, the variable x is evaluated to 8.
A. int x = 32;
   x = x >> 2;
B. int x = 33;
   x = x >> 2;
C. int x = 35;
   x = x >> 2;
D. int x = 16;
   x = x >> 1;

67. Which of the following represent legal flow control statements?
A. break;
B. break( );
C. continue outer;

Hidden page

75. Which of the following containers support the random access iterator?
    - A. priority-queue
    - B. multimap
    - C. list
    - D. vector
    - E. multiset
76. Which of the following are non-mutating algorithms?
    - A. search()
    - B. accumulate()
    - C. for_each()
    - D. rotate()
    - E. count()
77. Which of the following functions give the current size of a **string** object?
    - A. max_size()
    - B. capacity()
    - C. size()
    - D. find()
    - E. length()
78. Consider the following code:

```
class  Base
{
      private : int x;
       protected : int y;
};
class Derived : Public Base
{
      int a, b;
      void change()
      {
           a = x;
           b = y;
      }
};
int main()
{
       Base base;
       Derived derived;
       base.y = 0;
       derived.y = 0;
       derived.change();
}
```

Which of the lines in the above program will produce compilation errors?

7. What is the advantage of using named constants instead of literal constants in a program?
8. What is the difference between the following two declarations?

```
extern int m;
int m = 0;
```

9. How do the following two compare?

```
(a)  #define max(x,y) (((x)>(y) ? (x) : (y))
(b)  inline int max(int x, int y)
     { return (x>y) ? x : y; }
```

10. When the following code is executed, what will be the values of x and y?

```
int x=1, y=0;
y = x++
```

11. What are the values of m and n after the following two statements are executed?

```
int m=5;
int n=m++ * ++m;
```

12. Use type casts to the following statements to make the conversion explicit and clear.

```
float x = 10 + intNumber;
int m = 10.0 * intNumber/floatNumber;
```

13. What are lvalues and rvalues?
14. What are **new** and **delete**?
15. What is the difference between using **new** and **malloc()** to allocate memory?
16. In the following statements, state whether the functions **fun1** and **fun2** are value-returning functions or void functions.

```
(a)  x = 10 * fun1(m,n) + 5;
(b)  fun2(m,n);
```

17. What is the difference between using the following statements?

```
(a)  cin >> ch;
(b)  cin.get(ch);
```

18. Write a single input statement that reads the following three lines of input from the screen.

Hidden page

27. Given the statements

```
int y[5];
int *p = y;
```

is the following statement legal?

```
p[3] = 10;
```

28. How does a C-string differs from a C++ type string?
29. Does an array of characters represent a character string?
30. What is the difference between the following two statements?

```
const int M = 100;
#define M 100
```

31. Given the statement

```
const int size = 5;
```

can we declare an array as follows?

```
int x[size];
```

32. A character array **name** is defined as follows:

```
char name[30] = "Anil Kumar";
```

what will be the values of **m** and **n** in the following statements?

```
int m = sizeof(name);
int n = strlen(name);
```

33. Write a function **change()** to exchange to double values.
34. Write a function to sort a list of double values using the function **change()**.
35. What will be the value of **test** after the following code is executed?

```
int m = 10, n = -1, test = 1;
if(m<15)
      if(n>1)
         test = 2;
else
      test = 3;
```

Hidden page

42. Rewrite the following sequence of **if ... then** statements using a single **if ... then ... else** sequence.

```
if(m%2 == 0)
      cout << "m is even number \n";
if(m%2 != 0)
{
      cout << "m is odd number \n";
      cout << "m = " << m << "\n";
}
```

43. Simplify the following code segment, if possible.

```
if(value > 100)
      cout << "Tax = 10";
if(value < 25)
      cout << "Tax = 0";
if(value >= 25 && value <= 100)
      cout << "Tax = 5";
```

44. What does the following loop print out?

```
int m = 1;
while(m < 11)
{
      m++;
      cout << m++;
}
```

45. Write a code segment, using nested loops, to display the following output:

```
1  2  3  4  5
1  2  3  4
1  2  3
1  2
1
```

46. A program uses a function named **convert()** in addition to its **main** function. The function main declares a variable x within its body and the function **convert()** declares two variables **y** and **z** within its body, **z** is made static. A fourth variable **m** is declared ahead of both the functions. State the visibility and lifetime of each of these variables.

47. What is the output of the following program?

```
#include <iostream>
```

```
using namespace std;
void stat()
{
      int m = 0;
      static int n = 0;
      m++;
      n++;
      cout << m << "  " << n << "\n";
}
int main()
{
      stat();
      stat();
      return 0;
}
```

48. Replace **if ... else** ladder by a **switch** statement in the following code segment.

```
if(x == 5)
      a++;
else if(x == 6)
      b++;
else if(x == 9)
      c++;
```

49. What is the output of the following code segment?

```
int n = 0;
int i = 1;
do
{
      cout << i;
      i++;
}
while(i <= n)
```

50. What is the output of the following code segment?

```
int n = 0;
for(int i=1;i<=n;i++)
      cout << i;
```

51. Why is it inappropriate to use a **float** type variable as a loop control variable?

52. What is the output of the following statement?

```
cout<< "He \n said \n \" Hello \ " \n";
```

53. What is the primary purpose of C++ union types?
54. What are the two basic differences between a structure and an array?
55. Distinguish between a **struct** and a **class** in C++.
56. Name the three language features that characterize object-oriented programming languages.
57. What is the difference between static and dynamic binding of an operation to an object?
58. How would you write a generic version of **max** function that would return the largest of the two given values of any data type?
59. Compare the relationship between classes in composition and inheritance.
60. Distinguish between **virtual** functions and pure **virtual** functions.
61. Distinguish between static typing and dynamic typing.
62. What is the application of **reinterpret_cast** operator?
63. What is an abstract base class?
64. What is a pure **virtual** member function?
65. What is the application of **public**, **protected**, and **private** keywords?
66. Why do we declare some data members of a class as **private**?
67. Where and why do we need to use **virtual** functions?
68. What is dynamic binding? When do we use it?
69. What is a down cast? When do we use it?
70. Why do we need to use constructors?
71. What is a copy constructor? What is its purpose?
72. What is a default constructor?
73. What is '**this**'?
74. How are the overloaded operator functions useful in object-oriented design?
75. What is 'has a' relationship? How is this implemented?
76. What is 'is a' relationship? How is this implemented?
77. Will the following code work correctly?

```
void fun(int m)
{
     // code here
}
void fun(unsigned char m)
{
     // code here
}
int main()
{
     fun('X');
     return 0;
}
```

Hidden page

100. What is the use of the following code?

```
class  student
{
        static int m = 0;
        student()
       {
       m++;
       }
       .....
       .....
};
```

101. Which of the following expressions are wrong?

(a) `11% 2`
(b) `-11 % 2`
(c) `11 % -2`
(d) `-11 % -2`
(e) `11.0 % 2.0`

102. What will be the output of the following program segment.

```
{
       int m = 1;
       {
             int n = 2;
             cout << m << "   " << n << endl;
       }
       cout << m << "   " << n << endl;
}
```

103. What will be output of the following program?

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

bool test = false;
int main()
{
       bool test = true;
       cout << "test = " << test << "\n";
       cout << "test = " << :: test << "\n";
       return 0;
}
```

Hidden page

Hidden page

Hidden page

118. What is wrong with the following code?

```
class A
{
    protected: int x;
};
class B : public A
{
  public:
    void set(A a, int y)
    {
       a.x = y;
    }
};
```

119. What is the difference between a **set** and a **map**.
120. What is the difference between the C header **<string.h>** and C++ header **<string>**?

Bibliography

Balagurusamy, E, *Programming in ANSI C*, Tata McGraw-Hill, 1992.

Barkakati, Nabajyoti, *Object-Oriented Programming in* C++, SAMS, 1991.

Cohoon and Davidson, *C++ Program Design*, McGraw-Hill, 1999

Cox, B J and Andrew J Novobilski, *Object-Oriented Programming —An Evolutionary Approach*, Addison-Wesley, 1991.

Dehurst, Stephen C and Kathy T. Stark, *Programming in C++*, Prentice-Hall, 1989

Deitel and Deitel, *C++ How to Program*, Prentice Hall, 1998.

Eckel, Bruce, *Using C++*, Osborne McGraw-Hill, 1989

Gorlen K, *Data Abstraction and Object-Oriented Programming in C++*, Wiley, 1990.

Ladd S. Robert, *C++ Techniques and Applications*, M&T Books, 1990.

Lafore, Robert, *Object-Oriented Programming in Turbo C++*, Waite Group, 1999.

Lippman, Stanley B and Josee Lajoie, *C++ Primer*, Addison-Wesley, 1998.

Schildt, Herbert, *Using Turbo C++*, Osborne McGraw-Hill, 1990.

Stroustroup, Bjarne, *The C++ Programming Language*, 3rd edition, Addison-Wesley, 1997.

Stroustroup, Bjarne and Margaret A Ellis, *The Annotated C++ Reference Manual*, Addison-Wesley, 1990.

Voss, Grey, *Object-Oriented Programming —An Introduction*, Osborne McGraw-Hill, 1991.

Wiener, Richard S and Lewis J Pinson, *The C++ Workbook*, Addison-Wesley, 1990.

Index

Abstract base class 281
Abstract class 206
Abstract data types 9, 99
Accessing class members 101
Algorithms 402
Anonymous enum 41
ANSI prototype 102
ANSI standards 20, 446
ANSI C 20, 36
ANSI C++ 20, 36, 446
Argument counter 351
Argument vector 351
Arguments 79
Arrays 42, 109, 124, 166
Arrays of objects 124
Associative containers 405
Associativity 63
Asynchronous exceptions 381
AT&T C++ 30
Attributes 9
Automatic conversion 61
Automatic initialization 144

bad() 349
Basic data types 38
Binary Operators 176, 179
Bit fields 308
Bitwise expressions 60
bool 38, 60
Boolean expressions 59
Borland C++ 31
Bottom-up programming 4
Built-in data types 38

Call by reference 49, 81
calloc() 52
Cascading 27
Casting operator 190
Catch block 381
Catching mechanism 386
Chained assignment 60
char 45
cin 26
class 40, 99
Class dependency 484
Class hierarchies 485
Class templates 360
Class variables 121
Classes 8, 40, 96
Classification relationship 477
Client server 30
Client server relationship 479
cmath 91
Command-line arguments 350
Comments 21
Comparing strings 428
Compile time polymorphism 251
Compiling 30
Composition relationship 478
Compound assignment 61
Compound assignment
operator 61
const 41, 83, 132
const arguments 83
const objects 162
const_cast operator 450
Constant expressions 58
Constants 37
Constructors 144, 232, 326
Container adaptors 405
Containers 403

Containership 240
Containment relationship 484
Context diagrams 473
Control structures 64
Conversion function 190
Conversions 61, 172, 192
Copy constructor 149, 156
Copy initialization 156
cout 22
C-strings 428

#define 44
Data abstraction 8
Data dictionary 473
Data flow diagram 473, 483
Data hiding 8
Data members 8, 99
Decision table 473
Decision tree 473
Declaration statement 79
Declarations 45
Default arguments 84, 153
Default constructor 146
delete 49, 52
Dereferencing operator 52
Derived classes 201, 273
Derived containers 405
Derived data types 42
Destination class 191
Destructors 144, 162
Dot operator 97
double 38
do-while statement 68
Driver program 489
Dummy variables 79
Dynamic binding 10, 12, 252
Dynamic constructors 158
Dynamic initialization 46, 153
dynamic_cast operator 450

Early binding 251
Embedded assignment 60
Encapsulation 8, 99
endl 49, 55
End-of-file 334
Entry-controlled loop 68
enum 37
Enumerated data type 40
eof() 304
Error handling 348
Exception handling 380
Exception specification 392
Exceptions 380
Exit-controlled loop 68
explicit 451
Explicit call 147
Expressions 58
extern 45
Extraction operator 26

fail() 354
File mode 334
File pointers 335
Files 323
fill() 303, 307
Float expressions 59
Flowchart 5
for statement 68
Forest model 486
Fountain model 475
free() 52
Free store operators 53
Friendly function 89, 124, 179
fstream 325
Function objects 420,
Function overloading 10, 88
Function polymorphism 87
Function prototyping 79
Function templates 366
Functions 6, 42, 77
Fundamental data types 38

Generic programming 307
get() 293, 338
Get from operator 26
Get pointers 336
getline() 297
Global data 5
good() 349
Grid charts 472

header file 23
hierarchical char 478
hierarchical classification 8
hierarchical inheritance 202, 224

Identifiers 37
if statement 65
Implicit argument 270
Implicit call 147
Implicit constructor 152
Implicit conversions 61
Indirection 81
Information hiding 8
Inheritance 8, 12, 201–250
Inheritance relationship 477, 484
Initialization 46
Initialization list 238
Inline 82
Inline constructor 149
Inline functions 82–84, 149
Input operator 26
Input stream 291, 324
Instantiation 362
int 25
Integral expressions 59
Integral widening 62
iostream 20, 292, 325
ISO standard 20
Iterators 402, 408

Keywords 36

Late binding 252
Layout forms 472
Linking 28
Lists 413
Local classes 134
Local data 5
Logic errors 380
Logical expressions 60
long 38
Loop structure 63

main() 22
malloc() 52

Manipulators 54, 312
Mapped values 405
Maps 417
Math functions 90
math.h 90
Member function templates 373
Member functions 8, 101, 117, 132, 487
Memory management
    oprators 52
Message passing 10
Methods 8
Modifiers 38
Modular programming 4
multi-level inheritance 202, 214
multipath inheritance 229
Multiple inheritance 202, 218
mutable 452

Named constants 44
Namespace 25, 453
namespace 25, 453
Nesting 76, 241
Nesting of classes 240
Nesting of member functions 107
Nesting of Namespaces 454
new 49, 53

Object-based programming 12
Object-oriented analysis 474, 479
Object-oriented design 474, 483
Object-oriented languages 12
Object-oriented notations 475
Object-oriented paradigm 6, 473
Object-oriented programming 4, 6, 12, 474
Object-oriented system 468
Objects 6, 7, 29, 96, 101
open() 330
Operator function 172
Operator keywords 60, 459
Operator overloading 9, 23, 49, 63, 172,
    186
Operator precedence 63
Operators 49
Output operator 22
Output stream 291, 324

Overloaded constructors 151

Parameterized classes 359
Parameterized constructors 146
Parameterized functions 359
Pass-by-reference 122
Pass-by-value 122
Period operator 97
Play scripts 472
Pointer expressions 59
Pointers 43, 132, 251
Pointers to derived classes 273
Pointers to members 132
Pointers to object 253
Polymorphism 9, 251
precision() 301, 304
private 99
Private derivation 203
Procedure-oriented 4
programming Program flowchart 472
protected 211
Prototype paradigm 490
Prototyping 79, 103, 490
public 99
Public derivation 203
Pure virtual functions 281
put() 293, 338
Put pointer 336
put to operator 23

Random access 343
read() 339
References 47
Reference variable 47, 81
reinterpret_cast operator 385
Relational expressions 59
resetioeflags() 302
Return by reference 82
Return statement 25
Reusability 8, 201
Run time polymorphism 252

Scope resolution operator 49
seekg() 336
seekp() 336
selection structure 64
Sequence containers 404
sequence structure 64
Sequential access 338
set precision() 302
setf() 301, 307
setfill() 302
setiosflags() 302
setw 49, 54
setw() 50, 302
short 38
Shorthand assignment operator 61
Signed 38
Single inheritance 202, 204
Software crisis 1
Software evolution 3
Source class 191
Standard template library 401
static 45, 251
Static binding 251
Static class member 115
Static data members 115
Static linking 251
Static member functions 117
static_cast operator 449
Stream 291, 292, 326
Stream classes 291, 292
string class 428
string objects 428
struct 40
Structure chart 473
Structure elements 97
Structure tag 97
Structured programming 64
Structured 40, 96
swaping strings 438
switch 41
switch statement 67
Symbolic constants 44
Synchronous exceptions 381
Syntactic errors 380
System flow chart 472

tellg() 336
tellp 337

Template 81, 96, 359
Template arguments 374
Template class 362
Template function 368
Textual analysis 482
this pointer 270
Throw point 383
Throwing mechanism 386
Tokens 35
Top-down programming 4
Tree model 486
Try block 381
Turbo C++ 31
Type cast operator 57
Type compatibility 45
Type conversions 171, 187
typedef 58
typeid operator 451

Unary operators 173
union 40
UNIX 29
unsetf() 301
Unsigned 38
Use relationship 484
using 25

Variables 27, 45
Vectors 409
Virtual base class 228
Virtual function 89, 251, 252, 275
Visibility mode 203
Visibility modifier 210
Visual C++ 38
void 30

Warnier/orr diagrams 473
Waterfall mode 47, 470
wchar_t 39, 449
while statement 68
width() 301, 302
write() 287, 339

Hidden page

Hidden page

OBJECT ORIENTED PROGRAMMING WITH C++

FOURTH EDITION

The fourth edition of *Object Oriented Programming with C++*, explores the language in the light of its Object Oriented nature and simplifies it for novice programmers. The simple and lucid presentation of the concepts, the hallmark of this book, has been further enhanced in this edition.

Salient features:

- *Detailed coverage of Object Oriented Systems Development.*
- *Programming Projects – Two new projects on 'Menu Based Calculation System' and 'Banking System' for implementation.*
- *Step by step guidelines for implementation of projects.*
- *Model C++ Proficiency Test included to strengthen the concepts learnt in the book.*
- *Excellent pedagogy includes*
  - *84 Programming exercises*
  - *92 Solved programming examples*
  - *62 Debugging exercises*
  - *209 Review questions*

Dedicated Website: *http://www.mhhe.com/balagurusamy/oop4e*

visit us at www.tatamcgrawhill.com
ISBN :13 978-0-07-066907-9
ISBN :10 0-07-066907-4
9 780070 669079

Tata McGraw-Hill