

Then use these integers as the matrix indices to identify the cell to be opened. We have the `boardSet[5][4]` matrix containing the original values that needs to be displayed in the cell. Use the `gototxy` function to display the cell value in the respective position.

Replacing Unmatched Cells

When the user enters the input continuously, we have to match the cell values between two consecutive inputs. If the cell values are identical, then the cells have to remain visible. If not, the values in the opened cells needs to be replaced with the default values, that is, with the '@' symbol. This is enforced by the `ReplaceWithDefault` function. This function takes the cell numbers as its input and replaces the cell value with the default symbol in the same position if the cell values do not match.

Redrawing the Matrix

As the user enters the input continuously, the screen starts scrolling up and at a particular moment, the game board vanishes. At this stage, it becomes difficult for the user to enter the cell number without seeing the matrix. Hence, it is necessary to redraw the matrix after receiving the input from the user for a specific number of times.

To enforce this, a variable named "counter" is maintained. It is incremented whenever it encounters an output statement (including error messages). If this variable reaches a particular value the screen is cleared and the matrix is redrawn using the `CurrentMatrix` function. This function uses `boardCurrent[5][4]` matrix to store the current status of the game board.

Code for the Memory Game

```
/* MemoryGame.cpp */

#include "stdafx.h"
#include <iostream>
#include <string>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <windows.h>
#include "RandomGeneration.h"

using namespace std;

void gotoxy(short x, short y)
```

```
{
HANDLE hCon = GetStdHandle(STD_OUTPUT_HANDLE);
COORD pos;
pos.X=x-1;
pos.Y=y-1;
SetConsoleCursorPosition(hCon, pos);
}

class Game
{
    char boardDefault[5][4]; // board with @.
    char boardSet[5][4]; // board with values.
    char boardCurrent[5][4]; // current values held in the board.
    char cmp1,cmp2; // comparing the cell values.
    int tries; // No of attempts;
    int t1,t2; //holds the temp cell numbers
    int t3,t4; //holds the temp cell numbers
    int i;

public:
    Game()
    {

        i=0;
        tries=0;

    }

    void CreateBoard()
    {

        int k=0;
        cout<<"\n\n\n\n";
        cout<<"\t\t";

        // Printing the column headings.

        for(int j=0;j<4;j++)
        {
            cout<<"\t"<<j;
        }
        cout<<endl<<endl;
        cout<<"\t\t\t";

        // Underlining the column headings.
```

```
for( j=0;j<25;j++)
{
    cout<<"*";
}
cout<<endl<<endl;
cout<<"\t";

// Filling the board with the default characters

char row='A';
for(int i=0;i<5;i++)
{
    cout<<"\t"<<row;
    for(j=0;j<4;j++)
    {

        boardDefault[i][j] = '@';
        boardCurrent[i][j] =boardDefault[i][j] ;
        cout<<"\t"<< boardDefault[i][j];

    }
    cout<<endl<<endl;
    cout<<"\t";
    row++;
}

}

void FillBoard()
{

    char a[10];

    int randno[10];
    int rno;
    RandomGeneration *rg;
    rg=new RandomGeneration();

    rno=rg->DrawRandomNumber();
    //cout<<rno;

    for(int k=0;k<10;k++)
    {
        randno[k]=rno * k;
        a[k]=randno[k];
    }
}
```

```
        if(a[k]!='\0')
            a[k]='%';
        //cout<<a[k]<<endl;

    }

    // Filling the values for the board.
    // This symbol in the value gets opened when the user
    // enters the resepective cell number.
    k=0;
    for(int i=0;i<5;i++)
    {
        for(int j=0;j<4;j++)
        {

            boardSet[i][j] = a[k];
            //cout<<boardSet[i][j]<<"\t";
            if(k==9)
                k=0;
            else
                k++;

        }
        //cout<<endl;
    }

}

void ChangeCell()
{
    int bb,cnt=0;
    int counter =0;
    int cell_1=0,cell_2=0;

    char cellno[2];
    char choice;
    //cout<<endl;
```

do

```
{
    tries=0;
    cout<<"Enter -1 to exit ";
    CreateBoard();
    i=0;
    while(true)
    {

        i++;

        //gotoxy(1,22);

        int flag=0;
        for(int m=0;m<5;m++)
        {
            for(int n=0;n<4;n++)
            {
                if( boardCurrent[m][n] == '@')
                {
                    flag=1;
                    continue;
                }
            }
        }

        if (flag==0)
        {
            system("cls");
            gotoxy(24,9);
            cout<<" CONGRATULATIONS!!!!!" <<endl<<endl;
            gotoxy(26,10);
            cout<<"You have completed in "<<tries <<" Tries.";
            gotoxy(28,13);
            cout<<endl<<" PLAY AGAIN (Y/N) ";
            goto bb;
        }
        else
        {
cc: if(counter >2)
            {
                system("cls");
                CurrentMatrix();
            }
        }
    }
}
```

```
        counter =0;
        cnt=0;
        goto dd;
    }

dd:      cout<<"Enter the cell number to be opened: ";
        cin>>cellno;
        counter++;
        if(strlen(cellno)>2)
        {
            cout<<"\n Enter valid cell no.."; counter++; goto cc;
        }
        else
        {
            tries++;
            if (i==3)
            {
                if(cmp1 != cmp2)
                {

                    ReplaceWithDefault(t1,t2);
                    boardCurrent[t1][t2] = '@';
                    ReplaceWithDefault(t3,t4);
                    boardCurrent[t3][t4] = '@';
                    t1=t2=t3=t4=0;
                    i=1;

                }
            }
        }
        } //if
        else
        i=1;
    } //if
    if(cellno[0] == '-' && cellno[1] == '1')
    {
        system("cls");
        gotoxy(28,13);
        cout<<endl<<" PLAY AGAIN (Y/N) ";
        goto bb;
        break;
    }
    else
    { //cout<<cellno<<endl;

        // CONVERTING 'A','B','C','D','E' TO 0,1,2,3,4
```

```
switch(cellno[0])
{
    case 'A':
    case 'a':
        cell_1=0; break;
    case 'B':
    case 'b':
        cell_1=1; break;
    case 'C':
    case 'c':
        cell_1=2; break;
    case 'D':
    case 'd':
        cell_1=3; break;
    case 'E':
    case 'e':
        cell_1=4; break;
    default :
        {
            gotoxy(1,22+cnt);
            cout<<"\nEnter valid cell no..";
            //gotoxy(1,22+cnt);

            counter++;
            goto cc;// break;
        }
}
//cout<<cell_1<<endl;

//CONVERTING '0','1','2','3' TO 0,1,2,3
switch(cellno[1])
{
    case '0': cell_2=0; break;
    case '1': cell_2=1; break;
    case '2': cell_2=2; break;
    case '3': cell_2=3; break;
    default :
        {
            gotoxy(1,22+cnt);
            cout<<"\nEnter valid cell no..";
            counter++;
        }
}
```

```
        goto cc;
    }

}

if(boardCurrent[cell_1][cell_2] != '@')
{
    gotoxy(1,22+cnt);
    cout<<" Cell is already open... \n";
    goto dd;
}

//cout<<endl;
else
{
    IdentifyLocation(cell_1,cell_2,cnt);
    if (i==1)
    {
        cmp1 = boardSet[cell_1][cell_2];
        t1=cell_1;
        t2=cell_2;
    }
    if (i==2)
    {
        cmp2 = boardSet[cell_1][cell_2];
        t3=cell_1;
        t4=cell_2;
    }
    cout<<boardSet[cell_1][cell_2];
    // storing the current instance of the matrix

    boardCurrent[cell_1][cell_2] =boardSet[cell_1][cell_2];
    gotoxy(1,22+cnt); // to go to the next line
    cnt++;

} //else for re-entry of opened cell

} //else
} //else for strlen(cellno)
} //else
} //while
bb: cin>> choice;
    system("cls");
    } while(choice == 'Y' || choice=='y');
```



```
}

void CurrentMatrix()
{
    int i,j;

    cout<<"\n\n\n\n";
    cout<<"\t\t";

    for(j=0;j<4;j++)
    {
        cout<<"\t"<<j;
    }
    cout<<endl<<endl;
    cout<<"\t\t\t";

    for( j=0;j<25;j++)
    {
        cout<<"*";
    }
    cout<<endl<<endl;
    cout<<"\t";
    char row='A';
    for(i=0;i<5;i++)
    {

        cout<<"\t"<<row;
        for(j=0;j<4;j++)
        {

            if(boardCurrent[i][j] !='\0')
                cout<<"\t"<<boardCurrent[i][j];
            else
            {
                boardCurrent[i][j] = '@';
                cout<<"\t"<<boardCurrent[i][j];
            }
        }
        cout<<"\n";
        row++;
        cout<<"\n\t";
    }
}
}
```

```
void IdentifyLocation(int cell_1,int cell_2,int cnt)
{
    if(cell_1 == 0) // 'A'
    {
        switch(cell_2)
        {
            case 0: gotoxy(25,9); break;
            case 1: gotoxy(33,9);break;
            case 2: gotoxy(41,9); break;
            case 3: gotoxy(49,9);break;
            default :
            {
                gotoxy(1,22+cnt);
                cout<<"\nEnter valid cell no"; break;
            }
        }
    }
    //if k=0 , 'A'
    if(cell_1 == 1) // 'B'
    {
        switch(cell_2)
        {
            case 0: gotoxy(25,11); break;
            case 1: gotoxy(33,11);break;
            case 2: gotoxy(41,11); break;
            case 3: gotoxy(49,11);break;
        }
    }
    //if k=1, 'B'
    if(cell_1 == 2) // 'C'
    {
        switch(cell_2)
        {
            case 0: gotoxy(25,13); break;
            case 1: gotoxy(33,13);break;
            case 2: gotoxy(41,13); break;
            case 3: gotoxy(49,13);break;
        }
    }
    //if k=2 , 'C'
    if(cell_1 == 3) // 'D'
    {
        switch(cell_2)
        {
            case 0: gotoxy(25,15); break;
            case 1: gotoxy(33,15);break;
            case 2: gotoxy(41,15); break;
            case 3: gotoxy(49,15);break;
        }
    }
}
```

```
    }
    }//if k=3 , 'D'

    if(cell_1 == 4) // 'E'
    {
        switch(cell_2)
        {
            case 0: gotoxy(25,17); break;
            case 1: gotoxy(33,17);break;
            case 2: gotoxy(41,17); break;
            case 3: gotoxy(49,17);break;
        }
        // goto 11;
    }//if k=4 , 'E'
}

void ReplaceWithDefault(int val1,int val2)
{
    switch(val1)
    {
        case 0://{// 'A'
            switch(val2)
            {
                case 0: gotoxy(25,9); cout<<'@';break;
                case 1: gotoxy(33,9); cout<<'@';break;
                case 2: gotoxy(41,9); cout<<'@';break;
                case 3: gotoxy(49,9); cout<<'@';break;
            }
            break;
        }
        case 1://{// 'B'
            switch(val2)
            {
                case 0: gotoxy(25,11);cout<<'@';break;
                case 1: gotoxy(33,11);cout<<'@';break;
                case 2: gotoxy(41,11);cout<<'@';break;
                case 3: gotoxy(49,11);cout<<'@';break;
            }
            break;
        }
        case 2://{// 'C'
            switch(val2)
            {
                case 0: gotoxy(25,13);cout<<'@';break;
                case 1: gotoxy(33,13);cout<<'@';break;
            }
        }
    }
}
```

```
                case 2: gotoxy(41,13);cout<<'@';break;
                case 3: gotoxy(49,13);cout<<'@';break;
            }
            break;
        }
        case 3: { // 'D'
            switch(val2)
            {
                case 0: gotoxy(25,15); cout<<'@'; break;
                case 1: gotoxy(33,15); cout<<'@'; break;
                case 2: gotoxy(41,15); cout<<'@';break;
                case 3: gotoxy(49,15); cout<<'@';break;
            }
            break;
        }
        case 4: { // 'E'
            switch(val2)
            {
                case 0: gotoxy(25,17);cout<<'@';break;
                case 1: gotoxy(33,17);cout<<'@';break;
                case 2: gotoxy(41,17);cout<<'@';break;
                case 3: gotoxy(49,17);cout<<'@';break;
            }
            break;
        }
    } //outer switch

    } // ReplaceWithDefault();

};

void main()
{
    Game g;
    g.FillBoard(); // board with values
    g.ChangeCell();
}

/*RandomGeneration.h*/

//Generating random number

#include "stdafx.h"
#include <time.h>
```

```
#include <iostream>
#include<conio.h>

/*#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE

static char THIS_FILE[] = __FILE__;
#endif*/

using namespace std;

class RandomGeneration
{
    int Low,High;

public:
    RandomGeneration();
    void SetTimerSeed();
    int DrawRandomNumber();
    int GetHigh();
    int GetLow();

};

RandomGeneration :: RandomGeneration()
{
    Low = 1;
    High = 100;
}

// Set the seed to generate pseudo-random numbers. Use the current time & date as seed.

void RandomGeneration::SetTimerSeed() {
    time_t SeedTime;
    struct tm SeedDate;
    SeedTime = time(0);
    SeedDate = *localtime(&SeedTime);
    int FinalSeed = SeedTime + SeedDate.tm_mday + (SeedDate.tm_mon+1) +
(SeedDate.tm_year+1900);
    srand((unsigned int) FinalSeed);
}
```

```
// Return the Random number
// Interval : Self explanatory - Get the difference between the high and low Values
// RandomOffset = using the rand() function to generate random numbers from 0 to
Interval-1
// RandomNumber = Add the RandomOffset to the lowest number.

int RandomGeneration::DrawRandomNumber() {
    int Interval = GetHigh() - GetLow() + 1;
    int RandomOffset = rand() % Interval;
    int RandomNumber = GetLow() + RandomOffset;

    //cout<<RandomNumber;
    return RandomNumber;
}

// Standard Inpectors - Return High & Low values

int RandomGeneration::GetHigh() {
    return High;
}

int RandomGeneration::GetLow() {
    return Low;
}
```

A.2 The Employee Attendance System

Problem Statement

Create an application to track the employee leave information. The application should:

- Receive the employee information from the user.
- Validate the user entries such as name, address and phone number.
- Receive the attendance information for each employee.
- Accept the leave information, calculate the number of days leave taken and update the status of Casual Leave (CL), Medical Leave (ML) and Earned Leave (EL) accordingly.
- Display monthly attendance for all employees.
- Display the leave status for a particular employee or for all employees.
- Automate the generation of employee number and stores the complete employee information into a file.
- Calculate the leave based on the following table:

<i>Leave Mode</i>	<i>CL</i>	<i>ML</i>	<i>EL</i>
Max. Days Allowed per year	10	20	20

* Contain the following Main menu with options accessible using numeric keys:

<i>Main Menu</i>	<i>Key</i>
Enter Employee Data	1
Enter Leave Form	2
Enter Attendance	3
Close Monthly Attendance	4
Employee Leave Status	5
Exit	6

Learning Objectives

The designing of the Employee Attendance project enable the students to:

- Create a simple C++ application using the concept of Object-oriented programming
- Use pointers extensively
- Explore the use of date and time functions
- Perform validation to execute the program effectively
- Create, store, update and retrieve data in files

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.

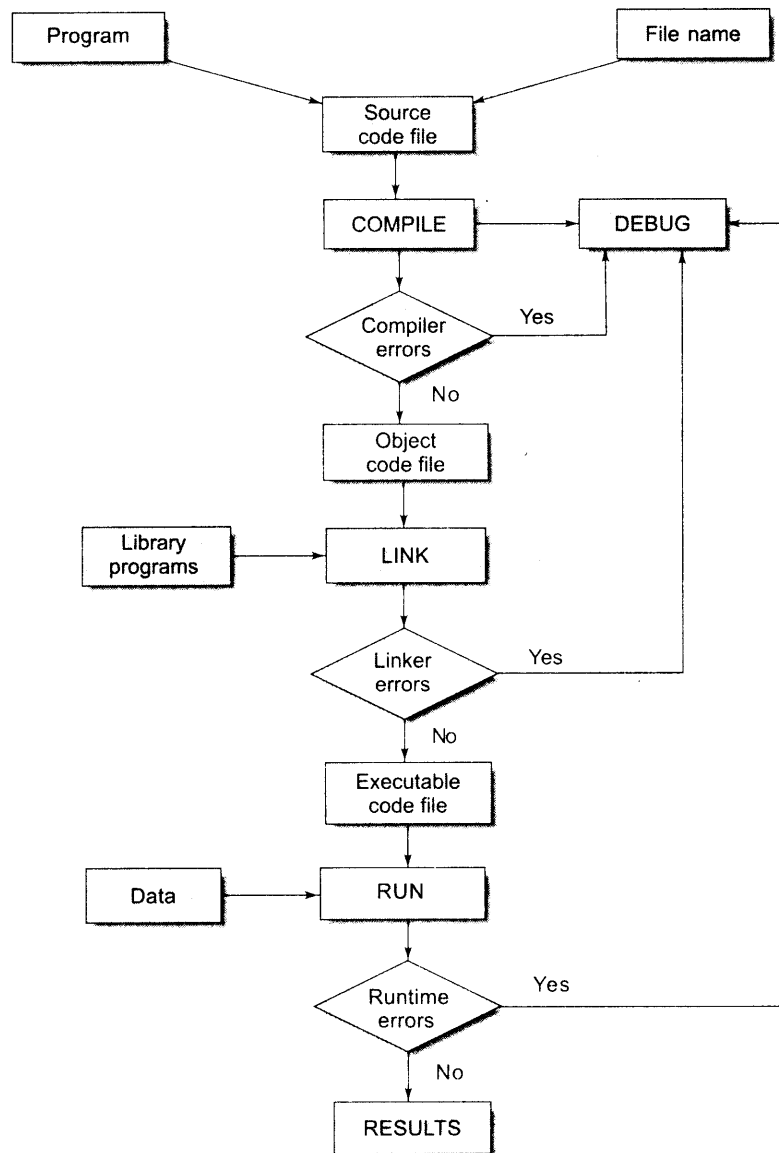


Fig. B.1 ⇔ Program development and execution

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)

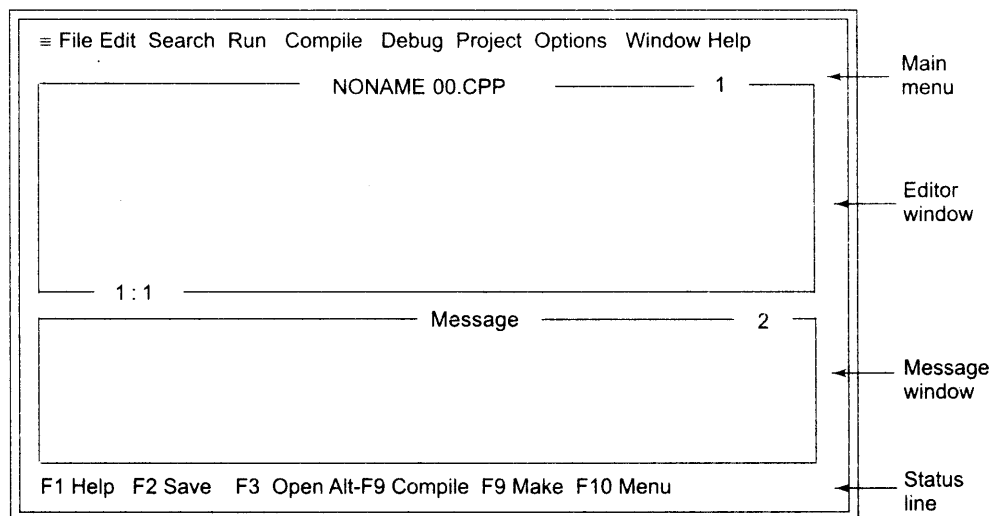


Fig. B.2 ⇔ IDE opening screen

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*

<i>Item</i>	<i>Options</i>
-	Displays the version number, clears or restores the screen, and execute various utility programmes supplied with Turbo C++
File	Loads and saves files, handles directories invokes DOS, and exists Turbo C+
Edit	Performs various editing functions
Search	Performs various text searches and replacements
Run	Compiles, 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.

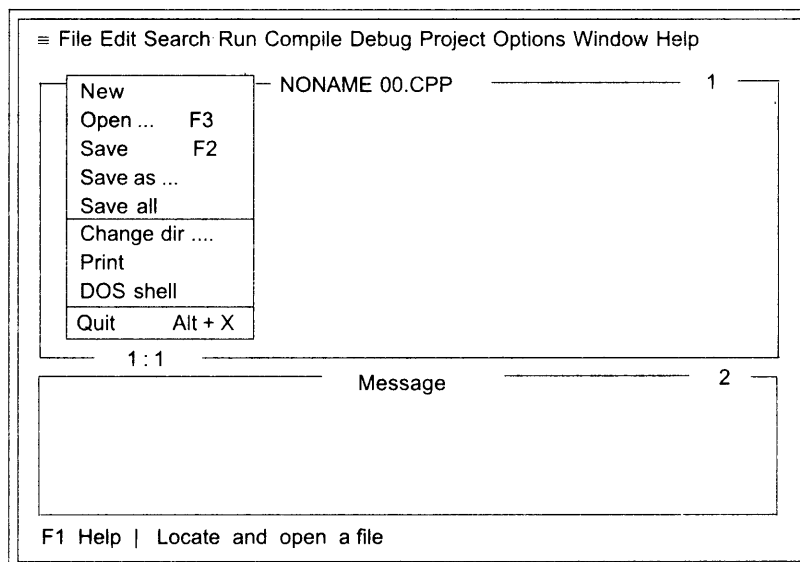


Fig. B.3 ⇔ *File dialogue window*

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.

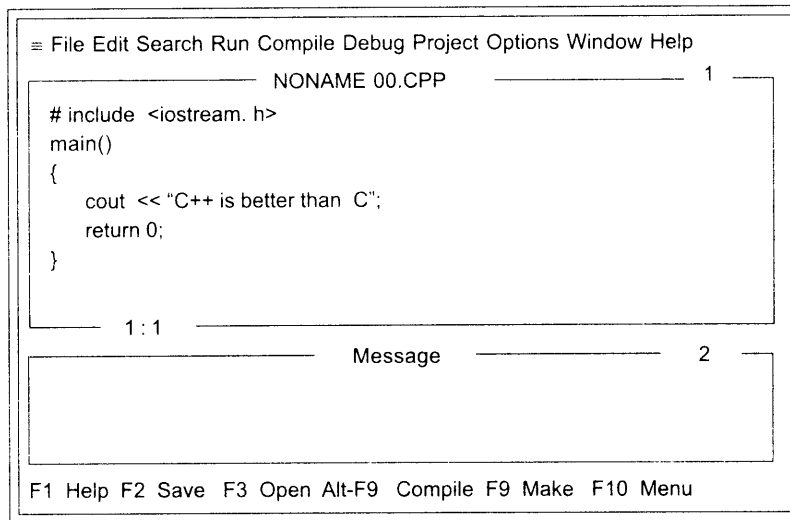


Fig. B.4 ⇔ Editor screen with statements

Once the typing is completed, you are ready to save the program in a file. At this time, you must go to the *File dialogue* menu again to select a suitable file option. Press F10 and select **File** option on the main menu. Select the **save as** option. This brings the save editor file window as shown in Fig. B.5.

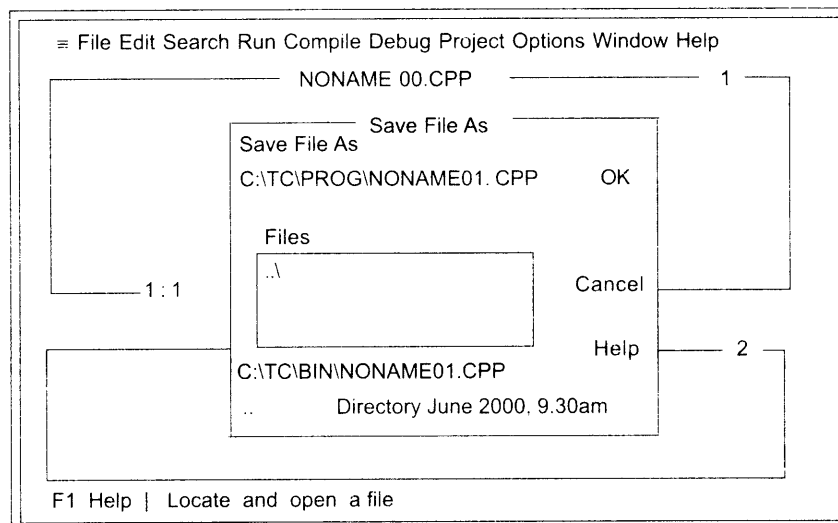


Fig. B.5 ⇔ Save editor file window

Now, you may change the file name NONAMEOO.CPP (shown in the editor file window) to the name you have selected. Make sure that your name has the extension **.CPP** to indicate to the compiler that your program is a C++ one and not C. Let's Assume that you have selected **test.cpp** as name. Press RETURN key and the program is saved in the file **test.cpp**.

B.7 Compiling the Program

When you select the **compile** option on the main menu, the *compile dialogue* window is displayed as shown in Fig. B.6. The *compile to OBJ* option allows you to compile the current file in the editor to an object file. In the present case, **test.obj** file is created, if there are no errors in your program. If there are any errors, appropriate error messages are displayed in the message window.

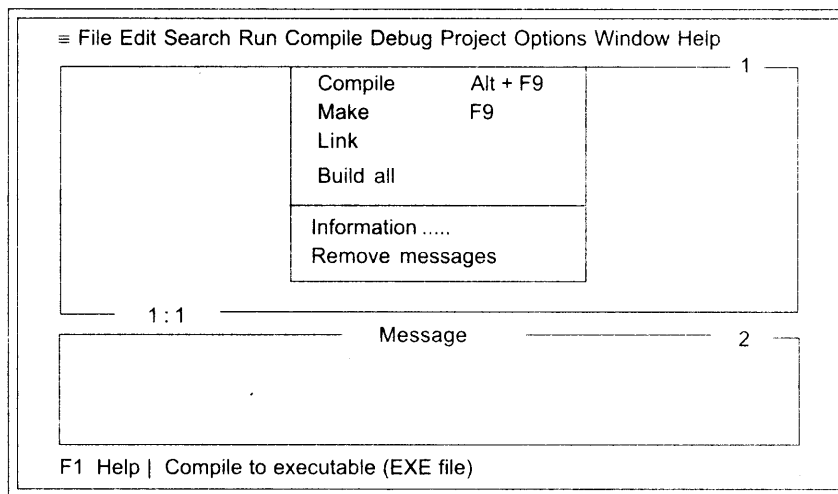


Fig. B.6 ⇔ *Compile dialogue menu*

During compilation, a window called *compilation window* will appear on the screen as shown in Fig. B.7. If there are no errors during compilation, this window will display “**Success: Press any key**” message. The entries for warning and errors will be 0.

B.8 Linking

To link the object file **test.obj** with the library functions, select the **LINK EXE** file option from the compile menu (See Fig. B.6). The **test.obj** will be linked and a third file named **test.exe** is created.

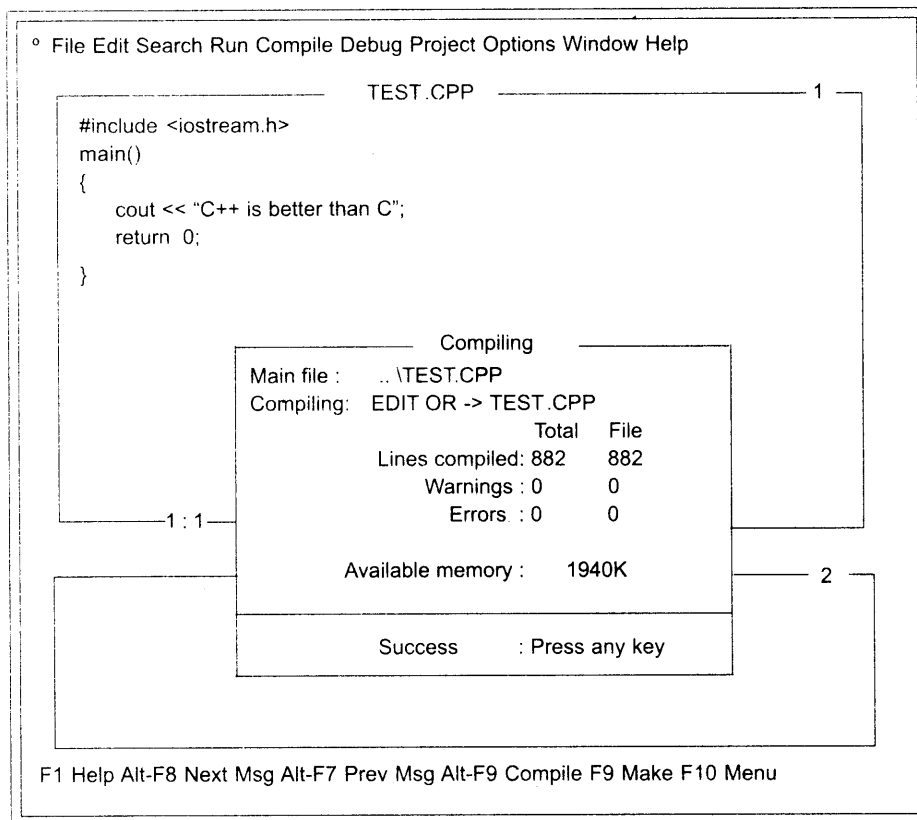


Fig. B.7 ⇔ *Compilation window*

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.

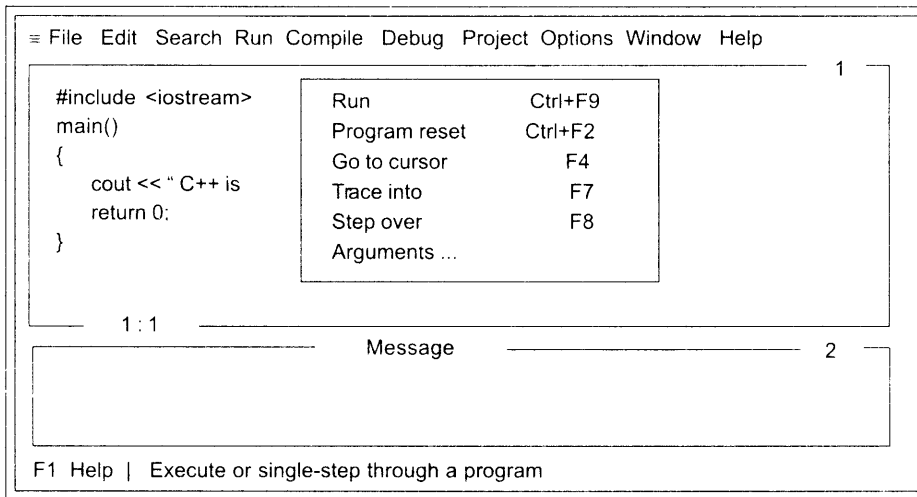


Fig. B.8 ⇔ Run dialog menu

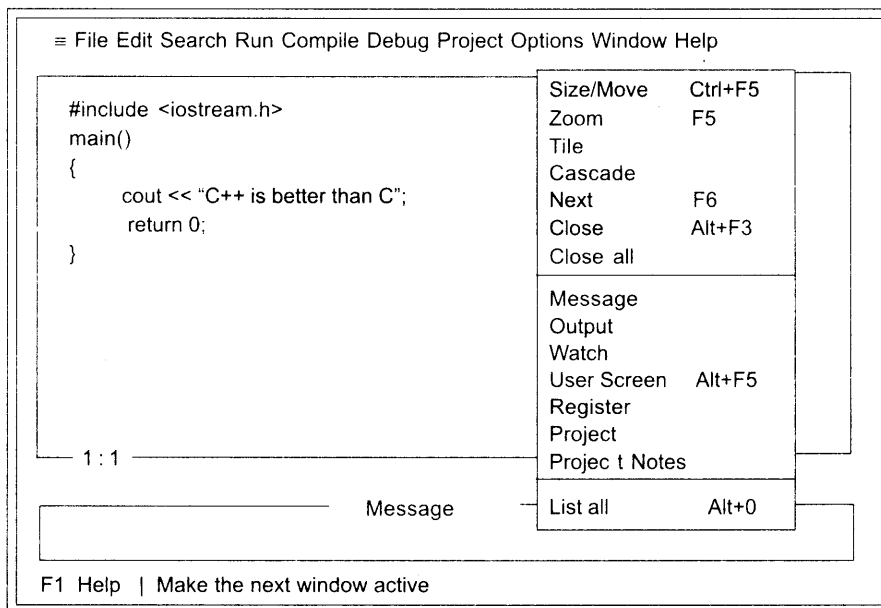


Fig. B.9 ⇔ Window dialog menu

B.10 Managing Errors

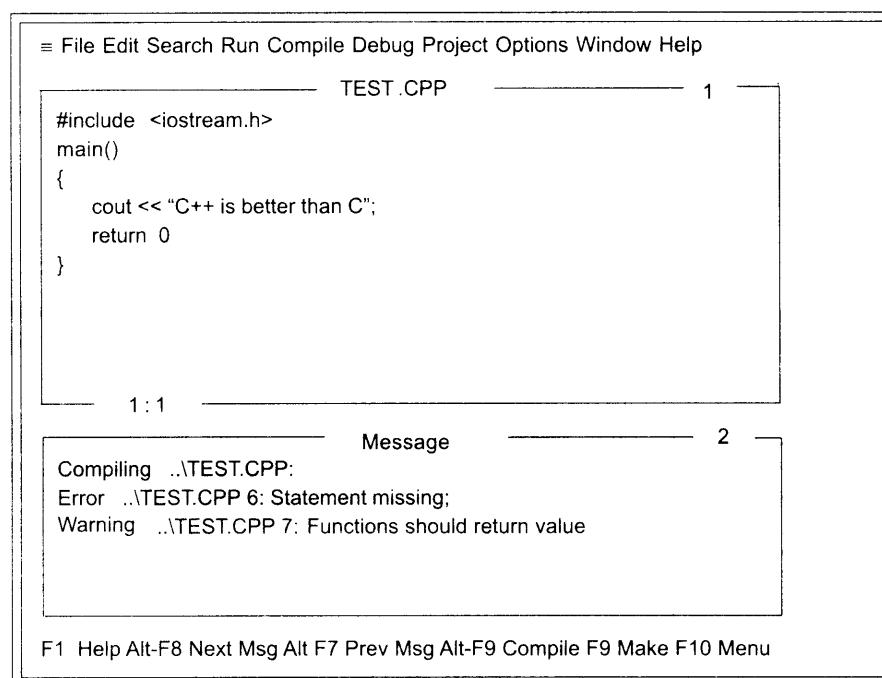
It is rare that a program runs successfully the first time itself. It is common to make some syntax errors while preparing the program or during typing. Fortunately, all such errors are detected by the compiler or linker.

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.



The screenshot shows a window titled "TEST.CPP" with a menu bar (File, Edit, Search, Run, Compile, Debug, Project, Options, Window, Help). The code in the editor is:

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

Below the code is a "Message" window showing the output of the compilation:

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

At the bottom of the window, there are keyboard shortcuts: F1 Help Alt-F8 Next Msg Alt F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu.

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	Executes 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)

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.

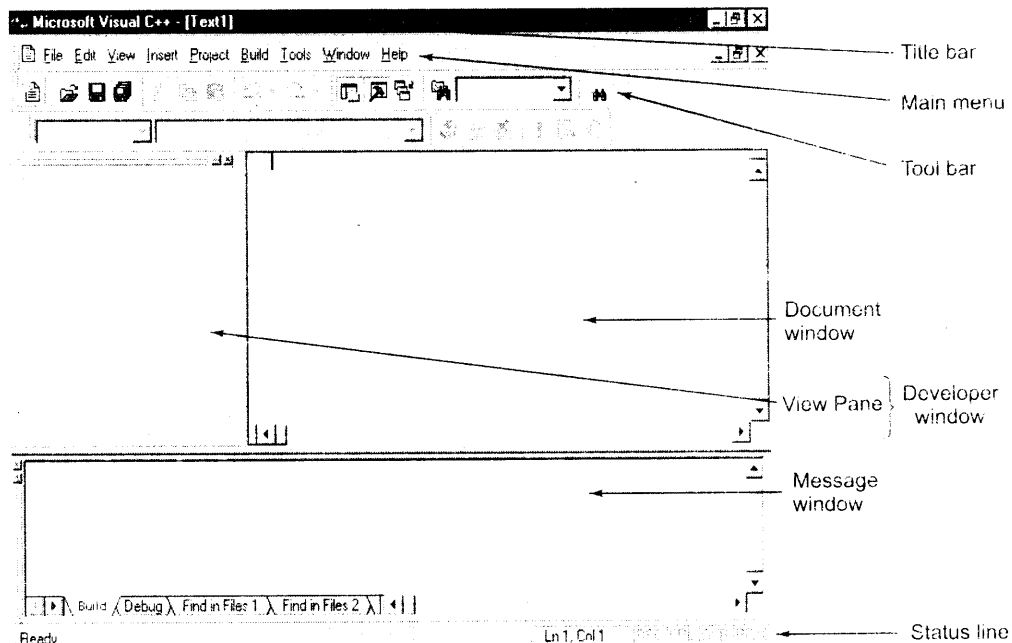


Fig. C.2 ⇔ Visual workbench opening screen

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

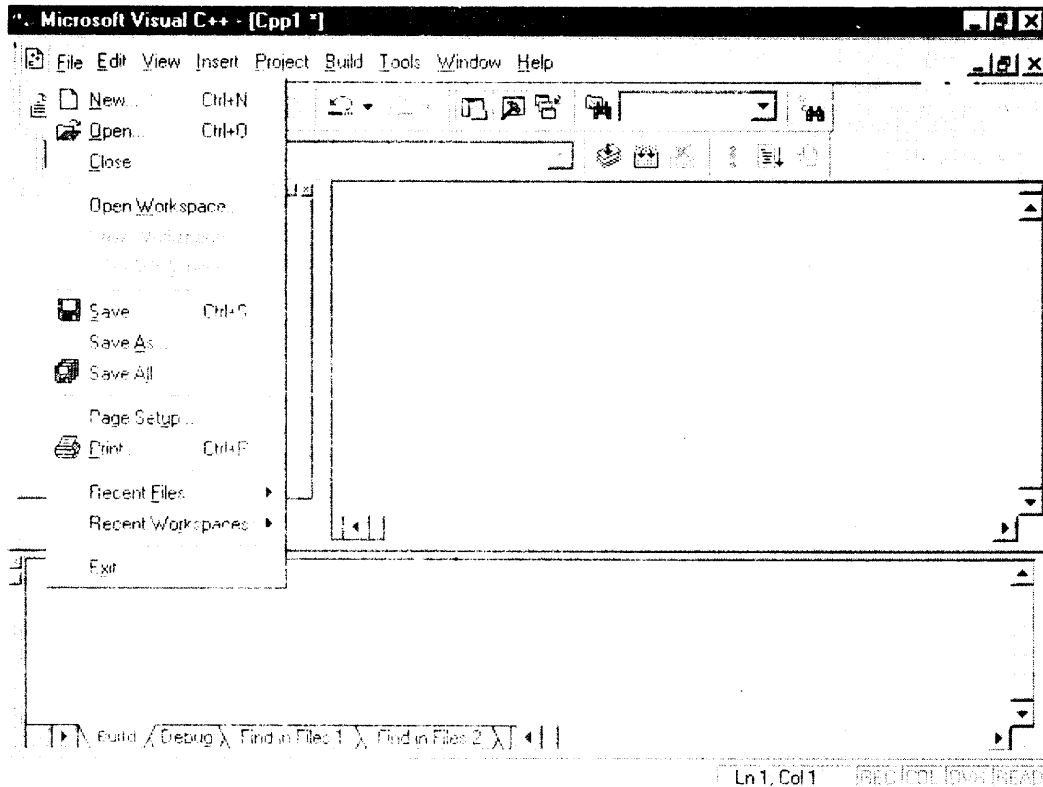


Fig. C.4 ⇒ Visual C++ Workbench file menu

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++ - [CPP1]' 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.

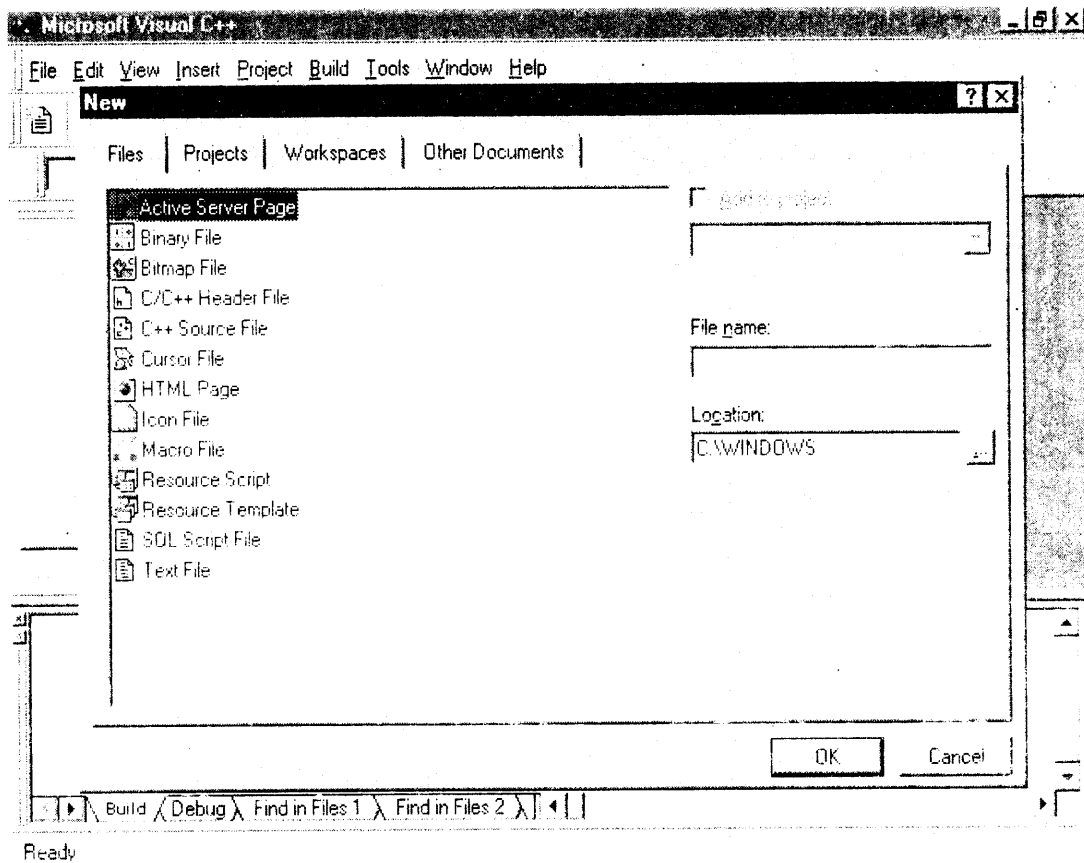


Fig. C.5 ⇔ *The new dialog box*

When a file is saved for the first time, the system will present you with a save dialog box. Save this under the file name TEST.CPP.

After the file is saved, the title in the title bar will show the saved file's name.

C.5 Compiling and Linking

In Visual C++, the process of compiling and linking all the source files to create an executable file is called “building”. There are three ways of compiling a type source code and is shown in Table C.2.

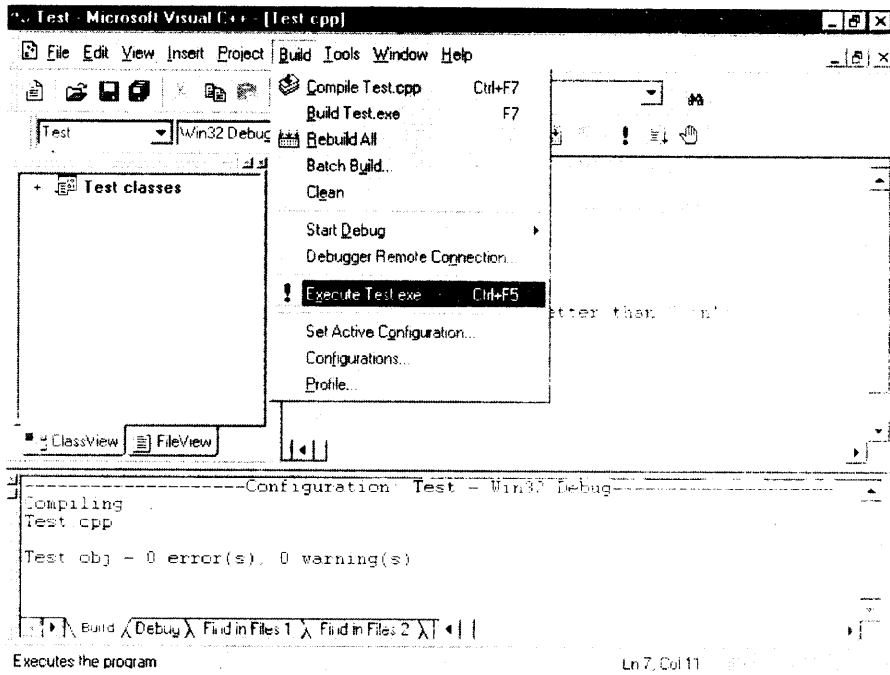


Fig. C.8 ⇔ Build menu after successful compilation

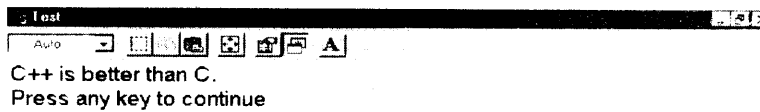


Fig. C.9 ⇔ Output generated

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.

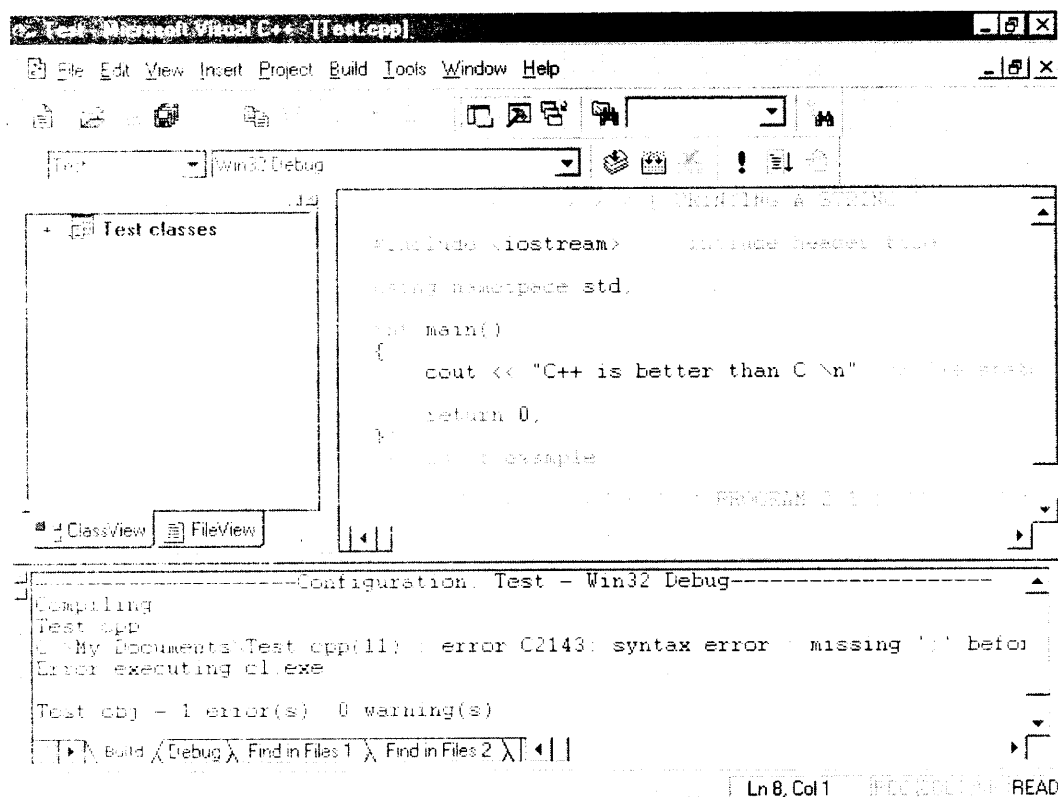


Fig. C.10 ⇔ *Output window error messages*

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

- 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 type 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	<p>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:</p> <pre> (i) if (expression) statement-1; statement-2; (ii) if (expression) statement-1; else statement-2; </pre> <p>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.</p>
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	<p>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</p> <pre> long int. long m; </pre>
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	<p>It is used to define a scope that could hold global identifiers. Example:</p> <pre> namespace name { Declaration of identifiers } </pre>
new	It is an operator used for allocating memory dynamically from free store. We can use new in place of malloc() function.

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 array		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
-=	subtract 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 modulus; and addition and subtraction.
36. It is a good programming practice to initialize a variable or an object when it is declared.