CHAPTER 6

GETTING STARTED WITH C++

INTRODUCTION

- ✓ C ++ Programming language was developed at AT & T Bells LAB early in 1980's by BJARNE STROUSTRUP.
- ✓ STROUSTRUP added some features to his favorite language Simula 67. (Earliest OOP).
- ✓ STROUSTRUP called " C with Classes ".
- ✓ C++ (C Plus Plus) is named by Rick Masitti.

BJARNE STROUSTRUP

Born	December 30, 1950) (age 60) Aarhus, Denmark
Occupation	College of Engineering Chair in Computer Science Professor, Texas A&M University USA

BJARNE STROUSTRUP

C++ CHARACTER SET

- ✓ Letters:- A-Z, a-z
- ✓ Digits:- 0 to 9
- ✓ Special Symbols:- space + - / ( ) [ ] = ! = < > , ‘ “ $ # ; : ? &
- ✓ White Spaces:- Blank Space , Horizontal Tab, Vertical tab, Carriage Return.
- ✓ Other Characters:- C++ can process any of the 256 ASCII Characters as data or as literal.

TOKENS

- ✓ The Smallest individual unit in a program is known as Token.
- ✓ Lexical Unit is the another name given to Token.
- ✓ C++ has following Tokens

1) Key words 2) Identifiers 3) Literals

4) Punctuators 5) Operators

KEY WORDS

- ✓ Key words are the words that convey special meaning to the compiler.
- ✓ Key words are also called as Reserved words meaning their meaning is reserved in C++.
- ✓ Some of Key words are,

  float int auto extern double

  case do while goto…etc

IDENTIFIERS

- ✓Identifier is an arbitrary long sequence of letters and digits.
- ✓The first character must be letter.
- ✓The underscore (_) counts as letter.
- ✓Upper and Lower case letters are different.
- ✓All characters are significant.

EXAMPLES OF IDENTIFIERS

- ✓ C++ is a Case Sensitive Language as it treats upper case letters and lower case letters differently.
- ✓ Some Valid identifiers,

  Myfile DATE9_7_8 z3t9x3
- ✓ MYFILE _DS _FXR

Some Invalid identifiers,

DATA-REC 28dre break

LITERALS

- ✓ Literals often referred to as constants

- ✓ These are the data items that never change their value during the program run.

TYPES OF C++ LITERALS

- ✓INTEGER – CONSTANT
- ✓CHARACTER – CONSTANT
- ✓FLOATING – CONSTANT
- ✓ STRING – LITERAL

INTEGER CONSTANT

- ✓ These are the whole numbers without any fractional part.
- ✓ An integer constant must have at least one digit and must not contain fractional part.
- ✓ It may contain + or - Sign.
- ✓ A number with no sign is assumed as positive.
- ✓ Commas can not appear in integer constant.

TYPES OF INTEGER CONSTANTS

- ✓DECIMAL INTEGER CONSTANTS
- ✓OCTAL INTEGER CONSTANTS
- ✓HEXADECIMAL CONSTANTS

DECIMAL INTEGER CONSTANT

- ✓ An integer constant consisting of sequence of digits is taken to be decimal integer constant unless it begins with 0.

  For instance 1234,56,+86,-56,-89,-22 are decimal integer constant.

OCTAL INTEGER CONSANT

- ✓A sequence of digits starting with 0(digit Zero) is taken to be an octal integer constant.
- ✓ for instance Decimal 8 is written as 010 as octal integer

$$8_{10} = 10_{8}$$

- ✓Decimal 12 is written as

$$12_{10} = 14_{8}$$

HEXADECIMAL INTEGER CONSTANT

- ✓ A Sequence of Digits preceded by 0x or 0X is taken to be an hexadecimal integer constant.
- ✓ For instance decimal 12 is written as
- ✓ $12_{10} = 0XC$
- ✓ The Suffix l or L and u or U is attached to any constant indicates as long and unsigned respectively
- ✓ For Example:- 12L,189U …etc

CHARACTER CONSTANT

- ✓A character constant is one character enclosed within single quotes

  Valid constants ‘a’ ‘z’ ‘k’

  Invalid Constants a z k
- ✓In c++ character constant must have single character and must be enclosed in single quotes.

NONGRAPHIC CHARACTERS

- ✓ Nongraphic character constants are those characters that can not be typed directly from the keyboard.

e.g backspace, tab, carriage return

- ✓ C++ Allows programmers to have certain nongraphic characters.

ESCAPE SEQUENCES

- ✓ Escape sequences are nongraphic character constants these are also called as non printing characters.
- ✓ These nongraphic characters are represented by means of escape sequence and is represented by a backslash (\ ) followed by a one or more character.

ESCAPE SEQUCENCE IN C++

Escape Sequence	Non Graphic Character
✓ \a	Bell
✓ \b	Back Space
✓ \f	Form Feed
✓ \n	New Line
✓ \r	Carriage Return
✓ \t	Horizontal Tab
✓ \v	Vertical Tab
✓ \’	Single Quote
✓ \”	Double Quote
✓ \?	Question Mark
✓ \on	Octal Number (On represents the number in octal)
✓ \xHn	Hexadecimal number
✓ \0	Null

NOTE

- ✓ THESE ESCAPE SQUENCES ARE USED IN OUT PUT STATEMENT COUT

FOR EX:

cout<<“\n\t Wel Come”; // Right statement

cin>>”\n\t”>>n; - Wrong statement

Do not use in cin statements

FLOATING CONSTANTS

- ✓ Floating constants are also called as Real constants. These may be written in one of two of forms called fractional form or exponent form.
- ✓ A real constant in fractional form must have one digit before a decimal point and at least one digit after the decimal point. It may also have either +ve or –ve sign preceding it. A real constant with no sign is assumed to be positive.

EXAMPLES OF REAL CONSTANTS

- ✓ The following are the valid real constants in fractional form.

  2.0 17.5 -89.9 -0.000732

- ✓ The following are the invalid real constants in fractional

  7 7. +27/3 24.34.32

12,45,344.09

REAL CONSTANTS IN EXPONENT FORM

- ✓ A Real constant in exponent form has two parts
  1. Mantissa
  2. Exponent
- ✓ Mantissa must be either an integer or proper real constant. The mantissa is followed by letter E or e and the exponent must be an integer.

EXAMPLE OF REAL CONSTANTS

- ✓ The floating point number 5.8 can be written as,
- ✓ $0.58 \times 10^{1} = 0.58E01$ or 0.58e01, here E01 or e01 represents 10^{1}

0.58	E01
Mantissa part	Exponent part

Valid Real Constants : 123E03, 0.76e02, -0.12e-2

Invalid Real Constants: 172.E03, 1.2E,0.4E2.3

STRING LITERAL

- ✓A String Literal is a sequence of characters surrounded by double quotes
- ✓ A Multi character constants are treated as string literals.
- ✓For Example: “abc” in the memory this string is represented as “abc\0” i.e along with the terminating character (Slash Zero).
- ✓String is an array of character.

PUNCTUATORS

Punctuators are also called as separators The Followings are used as punctuators

- ✓ Brackets [ ]
- ✓ Parantheses ( )
- ✓ Braces { }
- ✓ Comma ,
- ✓ Semicolon ;
- ✓ Colon :
- ✓ Asterisk *
- ✓ Ellipsis ...
- ✓ Equal Sign =
- ✓ Pound Sign #

OPERATORS

- ✓ Operators are tokens that triggers some computation when applied to a variable and other objects in an function.
- ✓ Unary Operators are those operators they act upon one operand.

OR

- ✓ operators that require one operator to operate upon.

UNARY OPERATORS

&	Address Operator
*	Indirection Operator
+	Unary Plus
-	Unary Minus
~	Bitwise Compliment
++	Increment Operator
--	Decrement Operator
!	Logical Negation

BINARY OPERATORS

Binary Operators are those operators that require two operands to operate upon.

- ✓ **Arithmetic Operators**

+ **Addition**

- **Subtraction**

***** **Multiplication**

/ **Division**

% **Remainder or Modulus**

SHIFT OPERATORS

<< (Shift Left)

>> (Shift Right)

For More information you will study in future chapters.

LOGICAL OPERATORS

&& Logical AND

|| Logical OR

ASSIGNMENT OPERATORS

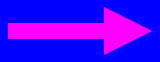
=	Assignment Operator
*=	Assign Product
/=	Assign Quotient
%=	Assign Remainder
+=	Assign Sum
-=	Assign Minus
<<=	Assign Left Shift
>>=	Assign Right Shift
&=	Assign Bitwise AND
^=	Assign Bitwise XOR
|=	Assign Bitwise OR

Relational Operator

<	Less than
>	Greater than
<=	Less than equal to
>=	Greater than equal to
==	Equal to
!=	Not equal to

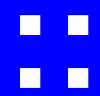
COMPONENT SELECTION OPERATORS

Direct component selector



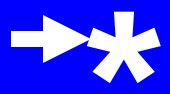
Indirect component selector

CLASS MEMBER OPERATORS



Scope access or scope resolution

Difference pointer to class member



Difference pointer to class member

CONDITIONAL OPERATOR

? :

Further Details you will study in

FLOW CONTROL CHAPTER

ORDER OF PRECEDENCE

A FIRST LOOK AT CPP PROGRAM

```
// My First Program
# include <iostream.h>
 int main ()
{
cout<<“Wel Come To C++ Programming”;
return 0;
}
```

The program produces the output as,

Wel Come To C++ Programming

FIRST PROGRAM

// My First C++ Program

This is the comment. All lines beginning with // are the comments in the program.

The comments are the non executable statements. When ever compiler identifies this is comment then it ignores

// is the single line comment

Starting with /* and ended with */ is used for multi line comments

FIRST PROGRAM

include <iostream.h>

Statement begin with # (hash or pound) sign are the directives for the preprocessor, that means these statement are processed before compilation takes place.

The # include <iostream.h> statement tells compiler's preprocessor to include the header file iostream.h in the program

FIRST PROGRAM

int main()

This line indicates the beginning of the main function. The main() function is the point where C++ program begin their execution.

return 0;

The return statement makes the main() to finish and it returns a value.

NOTE

- Every Executable statement in the C++ must be terminated by a semi colon

;

Why iostream.h ?

- The header file iostream.h is included in every C++ program to implement the input/output facilities.

  Input /output facilities are not defined in the C++ language rather they are implemented through a component of C++ standard library.

THE FUNTION OF I/O LIBRARY – iostream.h

1. At the lowest level, where the notion of data type is missing and the files are treated as streams of bytes, the I/O library manages the transfer of these bytes.
2. At the user level where the notion of data type is present, I/O library manages the interface between these two levels i.e between user level and the implementation level

PREDEFINED STREAMS IN I/O LIBRARY

- ✓ At the lowest level files are implemented as streams of bytes, then
- ✓ What is STREAM?

A Stream is simply a sequence of bytes.

Input and output operations are supported by the istream (input stream) and ostream (output stream) classes.

PREDEFINED STREAMS IN I/O LIBRARY

- ✓The ostream output operation, referred to as insertion and is performed by **insertion operator <<**

- ✓The istream input operation, referred to as extraction and is performed by **extraction operator >>**

EXTRACTION AND INSERTION SYMBOLS

- ✓ **extraction operator Symbol is,**

>>

- ✓ **insertion operator symbol is,**

<<

cin and cout statements

- ✓ cin (pronounced as see-in) stands for console input.
- ✓ cout (pronounced as see-out) stands for console output.

SYNTAX OF cin STATEMENT

The general format or the syntax of cin statement is,

cin>>variablename;

Or

cin>>variablename1>>variablename2>>variable name3……>>variablename*n;*

for example if four variables are to be input form the keyboard then,

cin>>a>>b>>c>>d;

Or the same statement can be written as follows,

cin>>a;

cin>>b;

cin>>c;

cin>>d;

SYNTAX OF cout STATEMENT

- **The general format or the syntax of cout statement is,**

cout<<variablename;

Or

cout<<variablename1<<variablename2<<variable name3……<<variablename*n;*

for example if four variables are to be input form the keyboard then,

cout<<a <<b <<c <<d;

Or the same statement can be written as follows,

cout<<a;

cout<<b;

cout<<c;

cout<<d;

cout STATEMENT

Another way of writing cout statement is,

cout<<string<<variablename;

where,

String is a sequence of characters enclosed within a double quotation.

Variable name is the value of variable which is to be printed after the message(string)

cout STATEMENT

totalamt;=6700;

cout<<“The Total Amount is = “<<totalamt;

Here, in this example

The Total Amount is= is the string

And the totalamt is the variable name, the value of the total is printed

Out put of this statement is as follows

The total Amount is = 6700

- ✓ As studied escape sequences are used in out put (cout) statements
- cout<<“\n\tBasic Pay=“<<basic<<“\n\t DA=“<<da<<“\n\t HRA = “<<hra<<“\n\t Gross Salary = “<<gs;

  output of the above statement is,

  Basic Pay=5900

  DA=40

  HRA =30

  Gross Salary = 10030

COMMENTS

- ✓ The comments are the non executable statements. When ever compiler identifies this is comment then it ignores. There are two types of comments in C++
- ✓ Single Line Comment (//)
- ✓ Multi Line Comment (starts with /* and ends with */

COMMENTS

The comments are the non executable statements. When ever compiler identifies this is comment then it ignores. Single Lien Comment Example

// My First C++ Program

This is the comment. All lines beginning with // are the comments in the program.

Starting with /* and ended with */ is used for multi line comments

Multi Line Comment Example

/* Write a CPP Program that calculates the sum and average of three numbers*/

I/O OPERATORS

- ✓ Input coming from the user's terminal referred to as standard input is tied to the predefined iostream cin and output directed to the user's terminal, referred to as standard output, is tied to the predefined iostream cout.

OUTPUT OPERATOR “<<”

The output operator (“<<”) (“put to”), also called as stream insertion operator is used to direct a value to the standard output device.

For example

```
cout<<“The Sum of 5 + 2 = “;
cout<< 5 + 2 ;
```

The statement will produce following output

The Sum of 5 + 2 = 7

OUTPUT OPERATOR <<

Now observe the output of following statement.

```
cout<< “The Sum of 5 + 2 is “;
cout<< 5 + 2;
cout<<“\n”;
cout<<“The Result pf 8 – 2 is“;
cout<<8 – 2 ;
```

The Output of abve set of statements will be as follows,

The Sum of 5 + 2 is 7

The Result pf 8 – 2 is 6

OUTPUT OPERATOR <<

The same statements can be written by using single cout statement

cout<< “The Sum of 5 + 2 is

<< 5 + 2<<“\n”

<<“The Result pf 8 – 2 is“<<8 – 2 ;

The Output will be same but code is reduced.

The Sum of 5 + 2 is 7

The Result pf 8 – 2 is 6

Note: Stream insertion operator >> signifies that the insert the value that follows in to the stream named cout.

/* Write a C ++ Program that prints the sum of two values */

```
#include <iostream.h> // for I/O Operation
#include<conio.h>// for clrscr() and getch() functions.
Int main()
{   cirscr() ;
    int value1, value2, sum ;
    cout << “Enter First Value:”;
    cin>>value1;
    cout<<“Enter Second Value:”;
    cin>>value2;
```

Program contd..

```
    sum=value1+value2;
    cout<<“The Sum of two given values is : “;
    cout<<sum;
    return 0;
}
```

Definition: Variable refers to a storage area whose contents can vary during processing. In this example sum , value1 , valu2 are the variables.

NOTE: The Stream extraction operator >>, signifies “extract the next value” from the stream named cin and assign it to the next named variable

OUTPUT OF PROGRAM

The output produced by the program is as follows,

Enter the First value: 6

Enter the Second value : 7

The Sum of the given value is : 13

CASCADING I/O OPERATOR

- The Multiple use of input or output operators (“>>” or “>>”) in one statement is called cascading of I/O operators.
- **For Example OUTPUT OPERATOR**

```
cout<<“The sum of “;
cout<<value1;
cout<< “and “;
cout<<value2;
cout<<“ is “;
cout<<value1+value2;
```

This can be using cascading of output operator as,

```
cout<<“The sum of “<<value1<< “and “<<value2<<“ is “
        <<value1+value2;
```

CASCADING I/O OPERATOR

- **For Example INPUT OPERATOR**

```
cout<<“Enter the two values “;
cin>>value1;
cin>>value2;
```

This can be using cascading of input operator as,

```
cout<<“Enter the two values “;
cin>>value1>>value2;
```

INVOKING TURBO C++

- To invoke (load) Turobo C++ Editor, Click on start and type cmd and press enter key. Go to root directory c and then type the following commands.

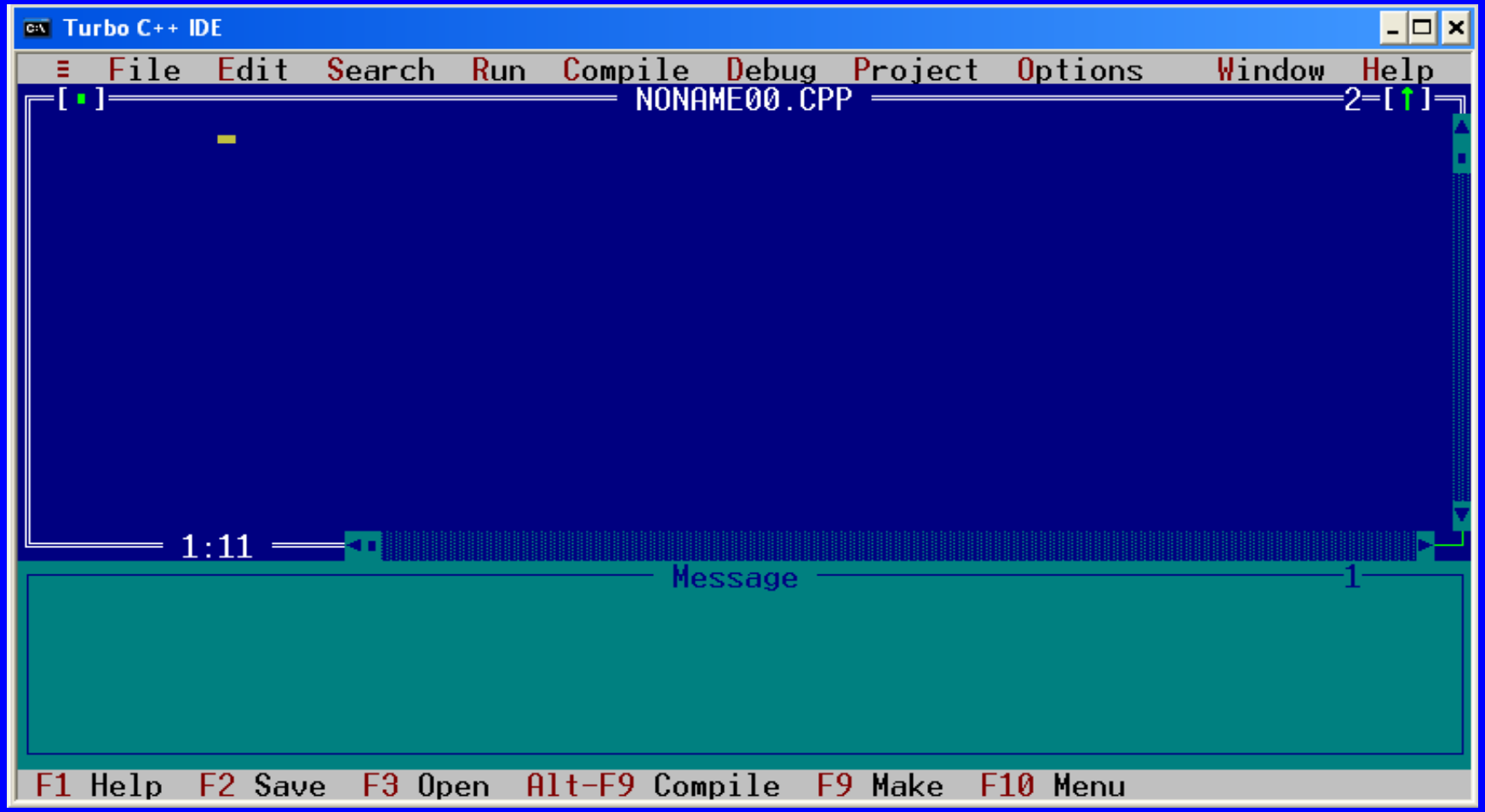
  C:\>cd tc

  C:\tc\>cd bin

  C:\tc\bin\>tc

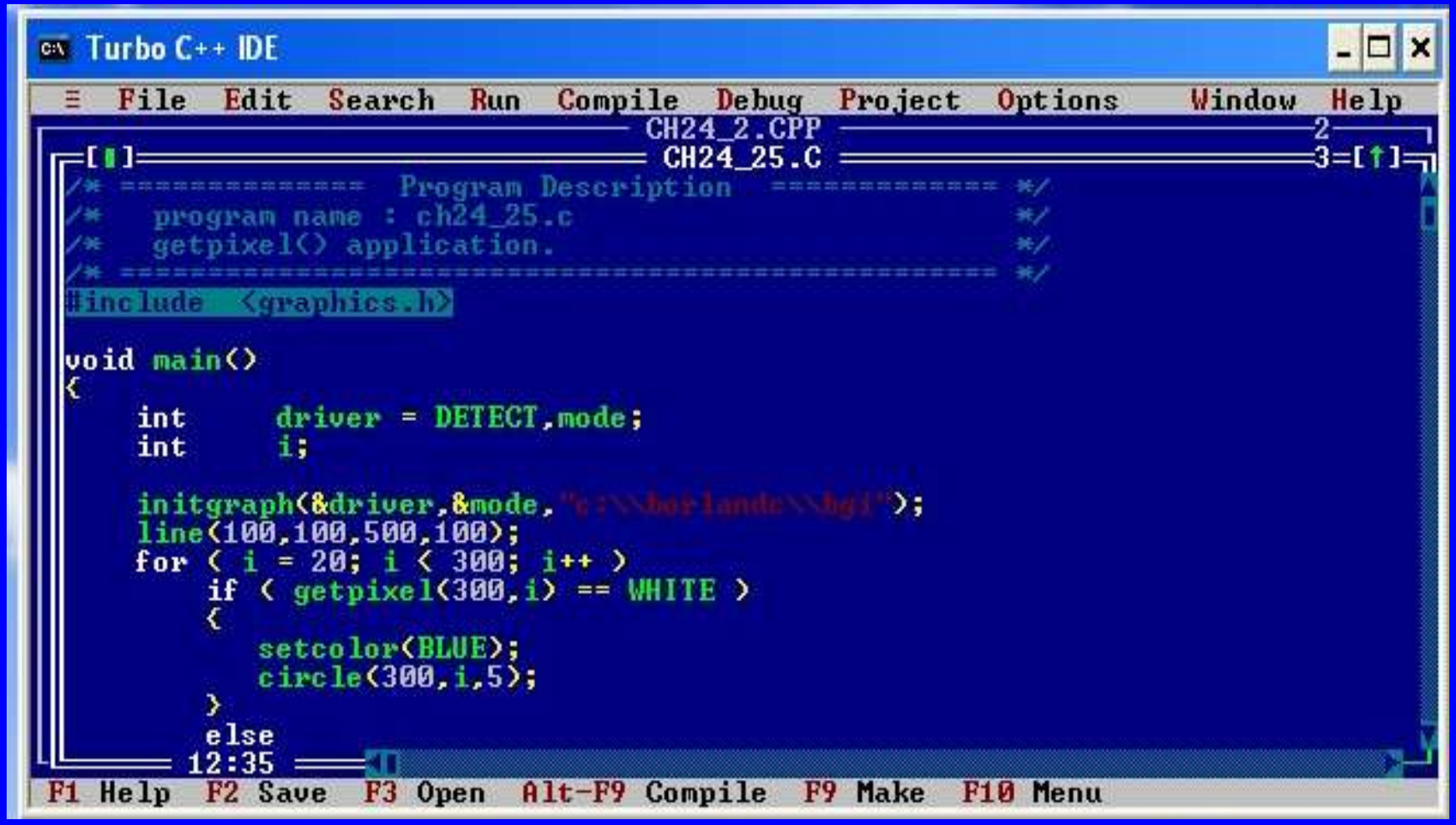
  After typing these commands at dos prompt you will see an C++ editor on the screen.

C++ EDITOR



Use F5 to Maximize Window and F6 to navigate between opened files

TYPING PROGRAM



COMPILING PROGRAM ALT + C or press ALT + F9

RUNING PROGRAM ALT + R or CTRL + F9

ROLE OF COMPILER

- A part of the compiler's job is to analyze the program code for **"Correctness"** if the meaning of the program is correct then compiler will not detect errors.

Types of Errors are,

1. **Syntax Errors**
2. **Semantic Errors**
3. **Type of Errors**
4. **Run Time Errors.**
5. **Logical Errors**

SYNTAX ERROR

Syntax errors refer to formal rules governing the construction of valid statements in a language.

Syntax errors occur when rules of a programming language are misused i.e., when a grammatical rule of C++ is violated.

1. SYNTAX ERROR

- For instance in the following program segment,

```
main()
{
  int a,b:
  cin>>a>>b;
  cout<<a+b,
  return 0;
}
```

2. SEMANTICS ERROR

- Semantics error occur when statements are not meaningful

  Semantics refers to the set of rules which give the meaning of the statement.

For example,

Sita plays Guitar

This statement is syntactically and semantically correct and it has some meaning.

2. SEMANTICS ERROR

See the following statement,

Guitar plays sita

is syntactically correct (syntax is correct) but semantically incorrect. Similarly, there are semantics rules of programming language, violation of which results in semantical errors.

X * Y = Z

will result in semantical error as an expression can not come on the left side of an assignment statement.

3. TYPE ERRORS

Data in C++ has an associated data type. The value 7, for instance, is an integer and ‘a’ is a character constant “Hi” is a string. If a function is given wrong type of data, then **Type Error** is assigned by compiler

For Example :

```
char  a[]=“Hi”;
pow(a,2);
```

This will result in type error.

4. RUNTIME ERRORS

A Run time error is that occurs during execution of the program. It is caused because of some illegal operation taking place.

For example

1. If a program is trying to open a file which does not exists or it could not be opened(meaning file is corrupted), results into an execution error.

2. An expression is trying to divide a number by zero are RUN TIME ERRORS.

5. LOGICAL ERRORS

- A Logical Error is that error which is causes a program to produce incorrect or undesired output.

  for instance,

```
ctr=1;
while(ctr>10)
{
    cout<<n *ctr;
    ctr=ctr+1;
}
```

MORE ABOUT COMPILER

- A Compiler reports an error by flashing an error message. An Error message contains a line number and a brief description of the error.

  A second thing, a compiler does is, it translates the corrected program text into ***object or assembly instruction text*** understood by the compiler, this process of translation is called **code generation.** After code generation actual program execution takes place.