20

Unit 2

BASICS OF PROGRAMMING OF C++

- Define computer program
- Describe the importance of syntax in any programming language

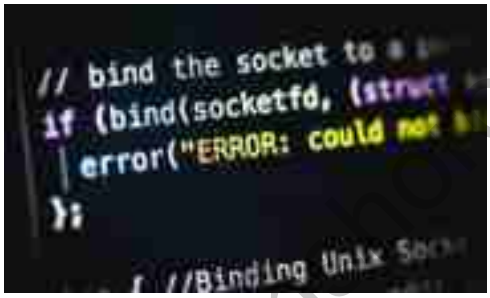
2.1 INTRODUCTION

A computer program is a set of instructions that is understood by a computer to perform tasks. A person who writes computer programs is known as a programmer. Computer processes instruction in binary language. Therefore, programs are written in programming languages. Programming languages have a specific set of words called syntax to create those instructions. Specialized programs such as compiler are used to convert set of syntaxes into set of machine-readable instructions. It requires an interface to convert commands from a human user. A programmer can make mistakes in syntax while writing a program or instructions. Other specialized programs, known as Integrated Development Environments (IDEs), help programmers to write programs in various languages. C++ is one of the most common and powerful general purpose programming language. All programming languages have certain concepts about rules of syntaxes, reserved words and data types.

2.1.1 Computer Program

A computer program is a collection of instructions that can be executed by a computer to perform a specific task. It is very difficult to write in the ones and zeroes of machine code, which is what the computer can understand, so computer programmers use specialized languages to communicate with computers to perform a set of specific tasks using languages like C++, Java or Python. Once it is written, the programmer uses a compiler to translate it into a language that the computer can understand.

Syntax in Programming Language:



Syntax tells the computer how to read a set of code. It is essentially a set of keywords and characters that a computer can read, interpret, and convert into tasks needed. Text-based computer languages are based on sequences of characters, while visual programming languages like Visual Basic are based on the layout and connections between symbols (which may be textual or graphical).

Example:

```
cout << "Hello World";
```

In C++, this syntax displays the message "Hello World" on the screen. Syntax plays an important role in the execution of programs in text-based programming languages and can even cause syntax errors if a programmer tries to run a program without using proper syntax. It is very common for new programmers to make syntax-based mistakes. Different programming languages use different types of syntax.

- Classify different programming languages into High, middle and low-level languages on the basis of their characteristics

2.1.2 Classification of Programming Languages

Thousands of programming languages have been developed till now, but each language has its specific purpose. These languages vary in how they can communicate with the computer's hardware. Some programming languages can directly communicate with hardware while others have less or no access to that

hardware. Based on the accessibility of hardware, they can be classified into following categories:

1. Low-level language
2. Middle-level language
3. High-level language

1. Low-level language

The low-level language is a programming language that can directly access and communicate with the hardware, and it is represented in 0 or 1 forms, which are the machine instructions. The languages that come under this category are the Machine level language and Assembly language.

Machine-level language:

The machine-level language comes at the lowest level in the hierarchy, so it has direct access to the hardware. It cannot be easily understood by humans. The machine-level language is written in binary digits, i.e., 0 and 1. It does not require any translator as the machine code is directly executed by the computer. Machine language is the first-generation programming language.

The assembly language:

The assembly language comes above the machine language means that it has lesser access to hardware. It is easy to read, write, and maintain by humans. The assembly language is written in simple English language, so it is easily understandable by the users. In assembly language, the assembler is required to convert the assembly code into machine code. It is a second-generation programming language.

2. Middle-Level Language

Some special purpose middle-level languages were developed in the past that were used as bridge between hardware and user interaction. However, such languages have become obsolete and are not used anymore.

3. High-Level Language

The high-level languages brought revolution in programming world. They allow a programmer to write the programs which are independent of a particular type of computer. These languages are closer to human languages than machine-level languages.

High-level languages do not have direct access to the hardware therefore a translator (compiler or interpreter) is required to translate a high-level language into a low-level language.

Advantages of a high-level languages

- The high-level language is easy to read, write, and maintain as it is written in English like words.
- The high-level language is portable as opposed to low-level languages; i.e., these languages are not dependent on the machine.

Differences between Low-Level language and High-Level language

Low-level language	High-level language
It is a machine-friendly language, i.e., the computer understands the machine language, which is represented in 0 or 1.	It is a user-friendly language as this language is written in simple English words, which can be easily understood by humans.
It requires the assembler to convert the assembly code into machine code.	It requires the compiler or interpreter to convert the high-level language instructions into machine code.
One type of machine code cannot run on all machines, so it is not a portable language.	The high-level code can be translated to required machine-code, so it is a portable language.
It has direct access to memory.	It is less memory efficient.
Coding and maintenance are not easy in a low-level language.	Coding and maintenance are easier in a high-level language.

- Distinguish among various types of translators

2.1.3 Translators

Computers only understand machine code (binary). This code is difficult to read, write and maintain. Programmers prefer to use a variety of high and low-level programming languages instead. A program written in any language is called as source code. To convert the source code into machine code, translators are needed.

A translator takes a program written in source language as input and converts it into a program in target machine language as output. It also detects and reports the error during translation.

Roles of translator are:

- Translating the program input into an equivalent machine language program.
- Providing alert messages wherever the programmer does not follow the rules of syntax of source language.

Different Types of Translators:

There are three different types of translators as follows:

1. Compiler

A compiler is a translator used to convert high-level programming language to low-level programming language. Compiler takes time to do its work as it translates high-level code to lower-level code all at once and creates an executable file. This translated program can be used again and again without the need for recompilation from source code.

It converts the whole program in one session and reports errors detected after the conversion. An error report is often produced after the full program has been translated. Errors in the program code may cause a computer to crash. These errors can only be fixed by changing the original source code and compiling the program again.

2. Interpreter

Interpreter is also a translator used to convert high-level programming language to low-level programming language. However, interpreter translates the code line by line and reports the error as soon as it is encountered during the translation process. With interpreter, it is easier to detect errors in source code than in a compiler. An interpreter is faster than a compiler as it immediately executes the code upon reading the code.

Interpreters do not create an executable file. Therefore, the interpreter translates the source code from the beginning every time it is executed.

3. Assembler

An assembler is a translator used to translate assembly language to machine language. It is like a compiler for the assembly language but interactive like an interpreter. An assembler translates assembly language code to an even lower-level language, which is the machine code. The machine code can be directly understood by the CPU.

- Differentiate between syntax, runtime and logical errors.

2.1.4 Types of Errors

Errors are the problems or the faults that occur in the program which cause the program to behave abnormally.

Programming errors often remain undetected until the program is compiled or executed. Some of the errors prohibit the program from getting compiled or executed. Thus, errors should be removed before compiling and executing.

The most common errors can be generally classified as follows:

1. Syntax Error

Syntax error occurs when the code does not follow the syntax rules of the programming language. These can be mistakes such as misspelled keywords, a missing punctuation character, a missing bracket, or a missing closing parenthesis. Nowadays, all famous Integrated Development Environments (IDEs) detect these errors as you type and underline the faulty statements with a wavy line. If you try to execute a program that includes syntax errors, you will get error messages on your screen and the program will not be executed.

Most frequent syntax errors are:

- Missing Parenthesis (})
- Printing the value of variable without declaring it
- Missing semicolon

2. Run-Time Error

Errors which occur during program execution (run-time) after successful compilation are called run-time errors. A run-time error occurs when a program is asked to do something that it cannot perform, resulting in a 'crash'. The widely used example of a run time error is asking a program to divide by 0.

The code contains no syntax or logic errors but when it runs it can't perform the task that it has been programmed to carry out.

3. Logic Error

Logic errors are those errors that prevent your program from doing what you expected it to do. On compilation and execution of a program, desired output is not obtained when certain input values are given. These types of errors which provide incorrect output but appear to be error free are called logical errors. These are one of the most common errors done by beginner programmers.

With logic errors you get no warning at all. For example, consider a program that prompts the user to enter three numbers, and then calculates and displays their average value. The programmer, however, made a logic error; one of its statements divides the sum of the three numbers by 5, and not by 3 as it should. The program will execute as usual, without any error messages, prompting the user to enter three numbers and displaying a result, but not the correct one. It is the programmer who has to find and correct the statement containing logical error.

- Discuss about Integrated Development Environment (IDE) of C++
- Develop the understanding about functions of different components of IDE

2.2 PROGRAMMING ENVIRONMENT OF C++

C++ runs on lots of platform like Windows, Linux, Unix, Mac, etc. Before we start programming with C++. We will need an environment to be set-up on our local computer to compile and run our C++ programs successfully.

2.2.1 Integrated Development Environment (IDE)

On a more basic level, IDEs provide interfaces for users to write code, organize text groups, and automate programming tools. Instead of a simple plain-text editor, IDEs combine the functionality of multiple programming processes into one. Most IDEs come with built-in translators. If any bugs or errors are found, users are shown which parts of code have problems.

Some IDEs are dedicated to a specific programming language or set of languages, having a set of tools and features which are helpful in writing codes for that language. For instance, Dev-C++ is used for making programs in C++ language. However, there are many multiple-language IDEs, such as Eclipse (C, C++, Python, Perl, PHP, Java, Ruby and more) and Visual Studio Code (Java, JavaScript, PHP, Python, Ruby, C, C++ and more).

Key Benefits of Integrated Development Environments:

- Serves as a single environment for most of a developer's needs such as compilation, linking, loading, and debugging tools.
- Code completion capabilities improve programming workflow.
- Automatically checks for errors to ensure top quality code.
- Refactoring capabilities allow developers to make comprehensive and mistake-free renaming changes.

2.2.2 Components of IDE

IDEs increase programmer productivity by combining common activities of writing software into a single application: editing source code, building executables, and debugging.

Editing Source Code

This feature is a text editor designed for writing and editing source code. Source code editors are distinguished from text editors because they enhance or simplify the writing and editing of code. Writing code is an

important part of programming. IDEs facilitate this process with features like syntax highlighting and autocomplete.

Syntax Highlighting

An IDE that knows the syntax of your language can provide visual cues. Keywords, words that have special meaning like class in C++, are highlighted with different colors. Syntax highlighting makes code easier to read by visually clarifying different elements of language syntax.

Code completion

When the IDE knows your programming language, it can anticipate what you're going to type next. Code completion features assist programmers by intelligently identifying and inserting common code components. These features save developers time writing code and reduce the chances of errors.

Compiler

Compilers are components that translate programming language into a form machines can process, such as binary code. IDEs provide automated build processes for languages, so the act of compiling and executing code is done automatically.

Linker

The linker opens the compiled program file and links it with the referenced library files as needed. Unless all linker items are resolved, the process stops and returns the user to the source code file within the text editor with an error message. If no problems encountered, it saves the linked objects as an executable file.

Loader

The IDE directs the operating system's program called the loader to load the executable file into the computer's memory and have the Central Processing Unit (CPU) start processing the instructions.

Debugging

No programmer can write programs without errors. When a program does not run correctly, IDEs provide debugging tools that allow programmers to examine different variables and inspect their code step by

step. IDEs also provide hints while coding to prevent errors before compilation. Programmers and software engineers can usually test the various segments of code and identify errors before the application is released.

2.2.3 Introduction to Dev-C++

One of the most commonly used IDE for coding programs in C++ is Dev-C++. It is a graphical IDE that has an integrated compiler system to create applications for Windows as well as console. Dev-C++ is a fully featured IDE supporting features like debugging, auto completion, localization, syntax highlighting, class and variable browsing, project management, package manager and others.

Installing and Configuring Dev-C++ IDE

Dev-C++ is freely available for download from this link:
https://sourceforge.net/projects/orwelldevcpp/

After downloading the installation package, we can begin the installation process. In this book, we will be using steps for installing Dev-C++ version 5.11 with the TDM-GCC 4.9.2 compiler.

Step 1.

Select "English" as the language to be used for installation process.

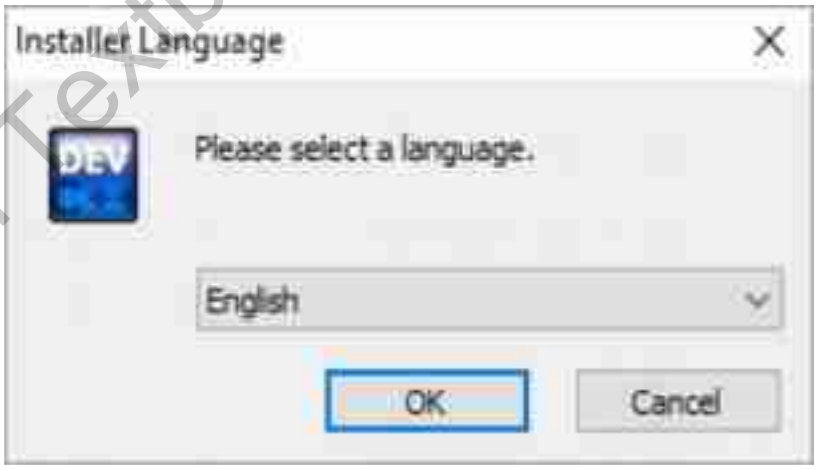


Fig. 2.1. Step 1: Dev-C++ installation

Step 2.

Agree to the license agreement by pressing "I Agree" button.

Step 3.

Select "Full" from the dropdown for "type of Install". This will select all the necessary components required to run Dev-C++ and compile C++ source codes. Click on "Next" to proceed.

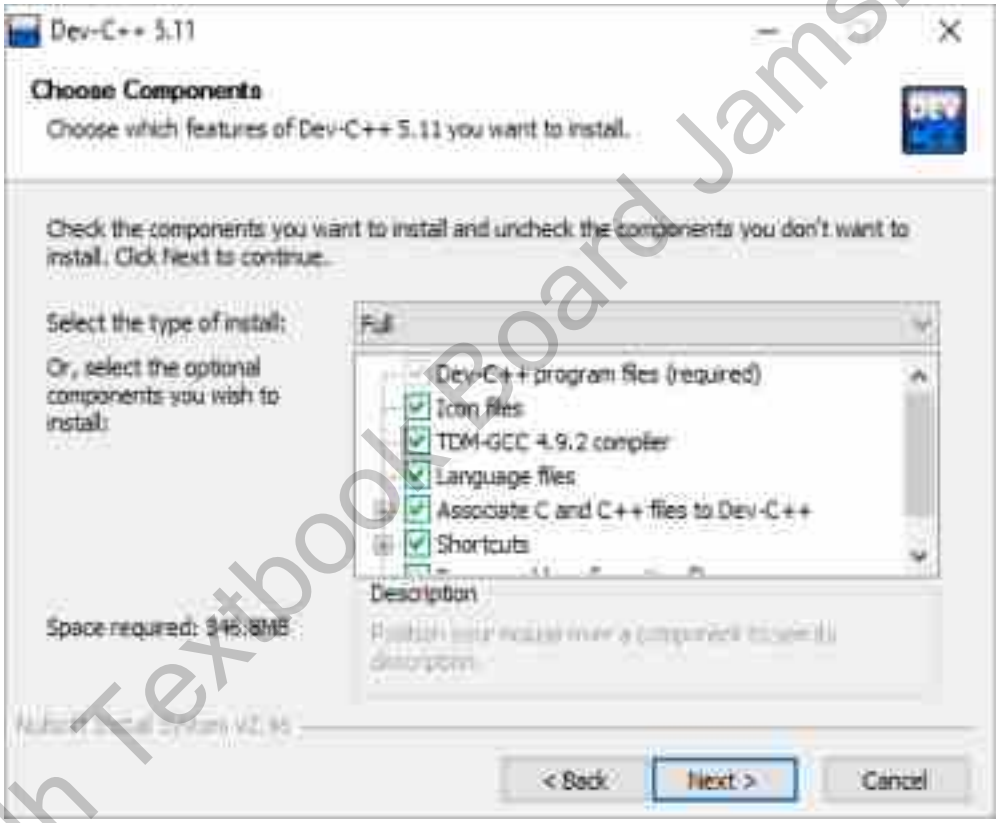


Fig. 2.2. Step 3: Installation components

Step 4.

Select the installation directory where all the necessary Dev-C++ files and libraries will be installed. Usually, the default specified path is used for installation but you can change it if desired. Click on "Install" to begin installation.

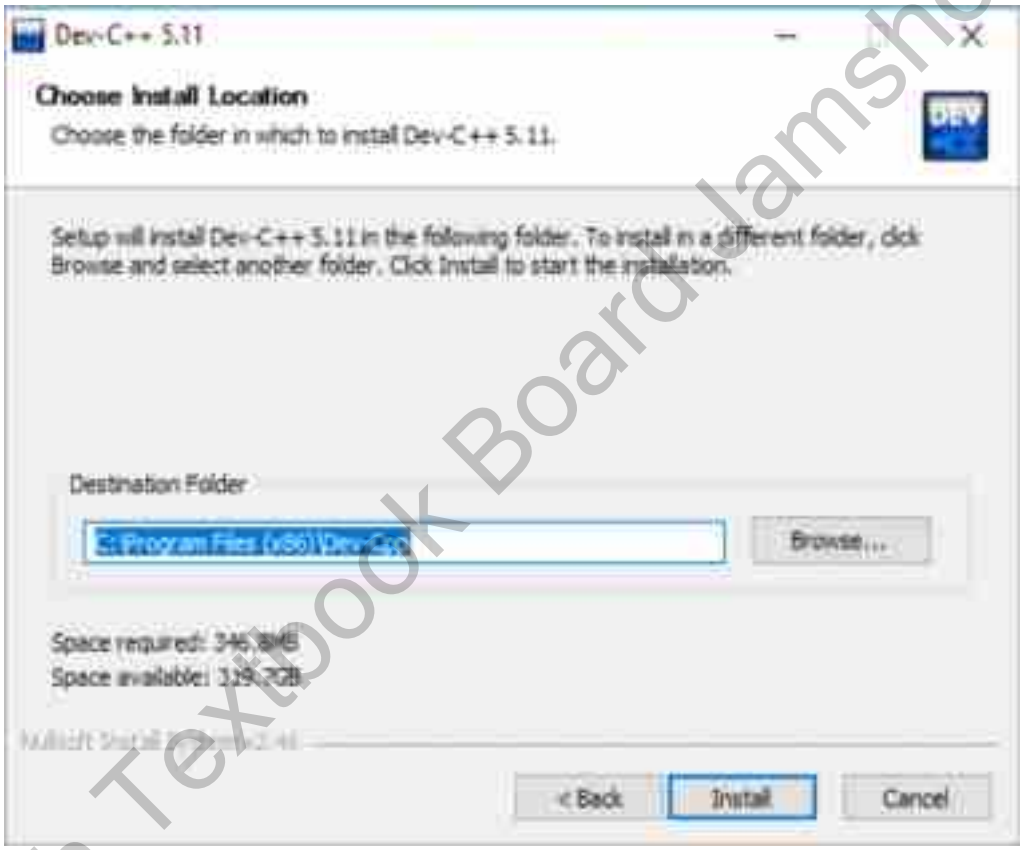


Fig. 2.3. Step 4: Install location

Step 5.

The installer will show the progress for installation. Once the process completes, it will show a "Finish" dialog. Make sure the "Run Dev-C++ 5.11" box is checked. This will automatically start Dev-C++ IDE after this installation completes. Click "Finish" button to complete the installation process.

Fig. 2.4. Step 5: Finish installation

Configuring Dev C++

When Dev-C++ IDE is run for the first time, it will require some configuration. This configuration will be used while developing programs in the IDE.

Set "English (Original)" as default interface language in the Dev-C++ first time configuration dialog. Click "Next" to continue. On the "theme" selection dialog, leave the default settings and click on "Next" to continue. Then click "OK" to close first time configuration dialog.

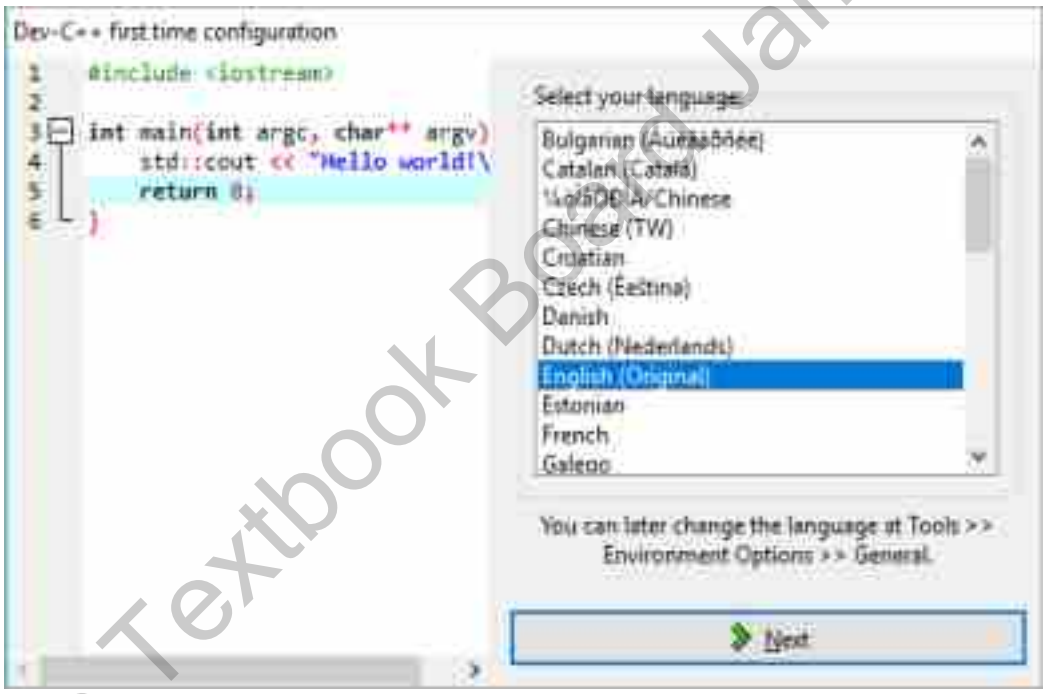


Fig. 2.5. Configuring Dev-C++

Linker Setting for Debugging

Sometimes an in-depth information is required from the debugger to properly identify the problems in our source code when a program is debugged. To obtain such information, our newly installed IDE and its integrated compiler needs to be configured. The following steps are used to enable this configuration:

1. Click on **Tools -> Compiler Options.**
2. Open the **Settings** tab from the Compiler Options dialog.
3. Under **Settings** tab, open **Linker** tab.
4. In the **Linker** tab, change the **Generate Debugging Information (-g3)** option to **Yes**.
5. Click on **OK** to save settings.

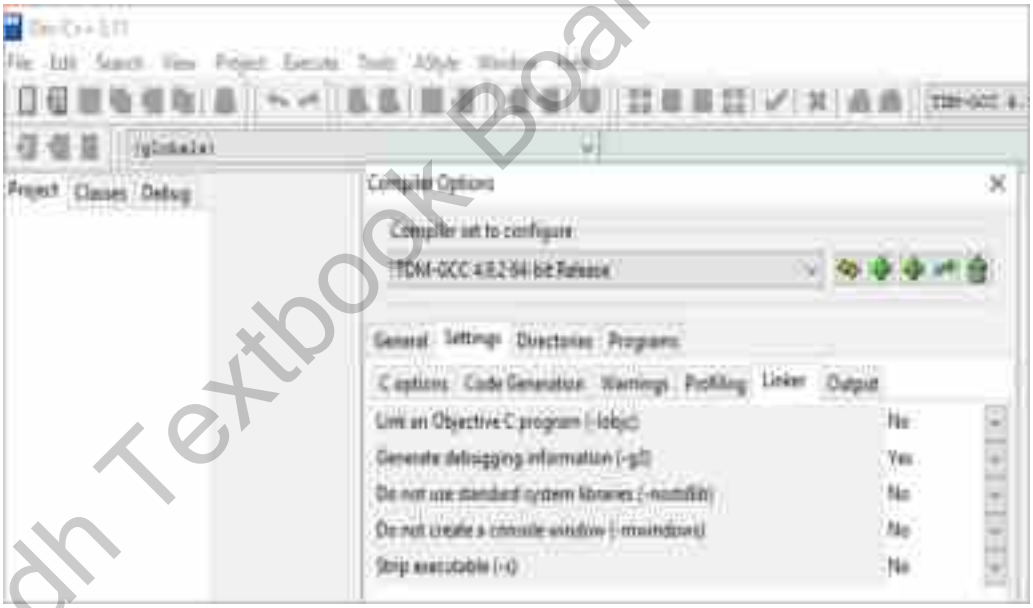


Fig. 2.6. Dev-C++ Computer options

Developing Programs in Dev-C++

C++ development is done by writing source codes and saving those files for compilation. Dev-C++ provides good project management support to help manage C++ files and group them into projects. The steps to create a new project in Dev-C++ are:

1. Click on **File -> New -> Project.**
2. From the **New Project** dialog, make sure **Empty Project** is selected. From language options, select **C++ Project**. Then enter a **Name** for your project.
3. Click on **OK**. Dev-C++ will ask for the path where you want the new project to be stored. Once it is done, Dev-C++ will open a workspace. It will show Project Explorer on the left side that shows the project we just created.

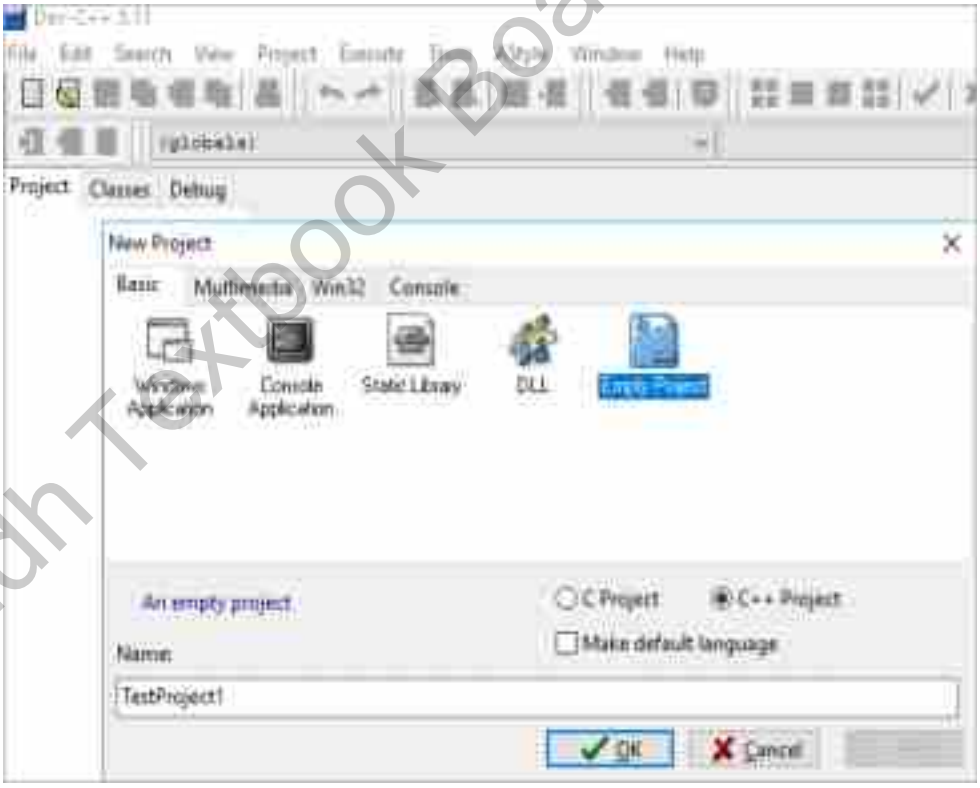


Fig. 2.7. Creating new project

Add Files to Project

A project requires source files which will contain codes for your program. The steps to create a new file are:

1. Click on **Project -> New File**. Alternatively, you can also right-click on the **Project Name** in the Project Explorer and click on **New File**.
2. Click on **Yes** on the Confirm dialog to add a file. This file is not stored until it is deliberately saved.
3. To save newly added file, click on **File -> Save**. Enter a path where you want to save the file and provide its name. Click on **Save** to store the file.

Compile and Execute Project

After writing the source codes in files, the project needs to be compiled and executed to see its output. Follow these steps to compile and run a project:

1. The project needs to be compiled before execution. To compile, click on **Execute -> Compile** or press **F9** key. **Compile Log** tab shows the compilation status. **Compiler** tab will show if there are any syntax errors.
2. After successfully compiling the project, run it by clicking on **Execute -> Run** or by pressing **F10** key.
3. A console window will open and show the output of the program.

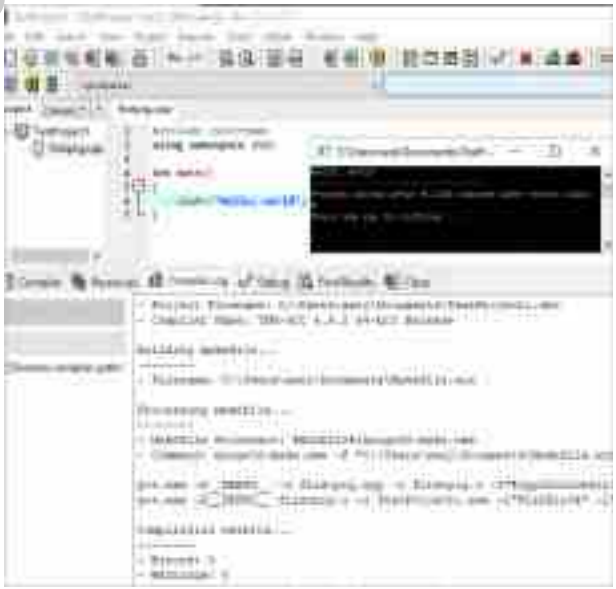


Fig. 2.8. Compile and execute project

- List out different reserved words commonly used in C++ program
- Use different data types in a C++ program

2.3 C++ PROGRAMMING LANGUAGE

C++ is a general-purpose programming language that was developed as an enhancement of the C language to include object-oriented concept. It was created by Bjarne Stroustrup and its main purpose was to make writing programs easier and more pleasant for the individual programmer.

C++ is a high-level language with an advantage of programming low-level (drivers, kernels) and even higher-level applications (games, GUI, desktop apps etc.). The basic syntax and code structure of both C and C++ are the same.

2.3.1 Reserved Words

A reserved word in C++ is a word whose meaning is already defined by the compiler. A reserved word cannot be used as an identifier, such as the name of a variable, function, or label – it is "reserved from use in C++". A reserved word is part of syntax and may not have any specific meaning in English language.

There is a total of 95 reserved words in C++. The reserved words of C++ may be conveniently placed into several groups. In the first group, we put those that were also present in the C programming language and have been carried over into C++. There are 32 of these.

There are another 30 reserved words that were not in C, are therefore new to C++ programming language. Some of the commonly used C++ reserved are:

and	break	case
auto	char	do
bool	class	else
catch	const	export
default	continue	float
double	delete	goto
enum	explicit	inline
extern	false	module
for	friend	new
if	import	or
int	long	protected
nullptr	namespace	short
public	not	static
requires	operator	struct
signed	private	template
switch	register	throw
this	return	typedef
true	sizeof	union
unsigned	try	virtual
void	using	while

2.3.2 C++ Data Types

You may need to store information of various formats and sizes like character, integer, floating point, double floating point, boolean etc. These formats and sizes are defined as data types. Based on the data type of a storage, the operating system allocates memory and decides what kind of data can be stored in that allocated memory.

C++ offers the programmer various types of built-in as well as user defined data types. Following table lists down some of the basic C++ data types:

Type	Keyword	Size	Range
Boolean	bool	1 byte	0 (false), 1 (true)
Character	char	1 byte	-127 to 127 or 0 to 255
Integer	int	4 bytes	-2147483648 to 2147483647
Floating point	float	4 bytes	1.5×10^{-45} to 3.4×10^{38}. Stores fractional numbers. Sufficient for storing 7 decimal digits
Double floating point	double	8 bytes	5.0×10^{-345} to 1.7×10^{308}. Stores fractional numbers. Sufficient for storing 15 decimal digits

- Differentiate between variable and constant
- Comprehend variable declaration rules in C++
- Differentiate between variable declaration and initialization

2.4 CONSTANTS AND VARIABLES

A **constant** is a value that cannot be altered by the program during execution, i.e., the value is constant. When associated with an identifier, a constant is said to be "named," although the terms "constant" and "named constant" are often used interchangeably. This is contrasted with a **variable**, which is an identifier with a value that can be changed during execution.

2.4.1 Constants and Variables

A constant is a data item whose value cannot change during the program's execution. Thus, as its name implies – the value is constant. Constants are used in two ways. They are:

1. literal constant
2. defined constant

A *literal constant* is a **value** you type into your program wherever it is needed. Examples include the constants used for initializing a variable and constants used in lines of code:

```
21, 12.34, 'A', "Hello world!", false, null
```

In addition to literal constants, there are symbolic constants or named constants which are constants represented by name. The `const` keyword and `#define` preprocessor are used to define a constant. Many programming languages use ALL CAPS to define named constants like `const float PI = 3.14159;` OR `#define PI 3.14159`.

A **variable** is the memory location that can hold a value. This value can change during the program's execution. It does not remain constant. For example, a classroom with a capacity of 20 students is a fixed place or constant but the subjects taught, teachers and students will vary with each class and subject and are variables.

Variables do not require to be assigned initial values. Variables once defined may be assigned a value within the instructions of the program. Variable can be assigned different values at different times during execution. For example:

```
x = 5;
x = 37;
```

Difference between Constant and Variable:

Constant	Variable
A constant does not change its value during program execution.	A variable, on the other hand, changes its value depending on instructions.
Constants are usually written in numbers and may be defined in identifiers.	Variables are always written in letters or symbols.
Constants usually represent the known values in an equation, expression or in line of programming.	Variables, on the other hand, represent the unknown values.

2.4.2 Rules for Naming Variables

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits and underscores
- Names must begin with a letter or an underscore (_)
- Names are case sensitive (myVar and myvar are different variables)
- Names cannot contain whitespaces or special characters like !, #, %, etc.
- Reserved words (like C++ keywords, such as int) cannot be used as names
- Names cannot be longer than 32 characters in C++ by default.

2.4.3 Declaring (Creating) and Initializing Variables

In C++, there are different **types** of variables (defined with different keywords). A variable declaration tells the compiler where and how much storage to create for the variable. A variable declaration specifies a data type and name for that variable as follows:

Syntax

```
data_type variable_name;
```

Where *type* is one of C++ data types (such as int), and *variable_name* is the name of the variable (such as **x** or **myName**).

Initialization

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

Syntax

```
data_type variable_name = value;
```

The **equal sign** is used to assign values to the variable.

Strings in C++

Variables that can store non-numerical values that are longer than one single character are known as strings.

The C++ language library provides support for strings through the standard string class. This is not a fundamental type, but it behaves in a similar way as fundamental types do in its most basic usage. Strings can be declared without an initial value and can be assigned values during execution.

- A computer program is a list of instructions that tell a computer what to do.
- We refer to syntax in computer programming as the concept of giving specific word sets in specific orders to computers so that they do what we want them to do.
- Different programming languages can be classified into high, middle and low-level languages.
- High-level languages are easy to read for humans and contain English language like words.
- Middle-level languages have a human readable format along with direct control over the machine's resources.
- Low-level languages are easy for machines to read and hard for humans. Low-level programs mostly comprise of binary digits and memory operators.
- There are three types of translators namely, compilers, interpreters and assemblers.
- Compilers convert high-level languages into machine readable format.
- Interpreters also convert high-level programs into machine readable format.

- Unlike compilers, interpreters convert instructions line-by-line.
- Assemblers convert low-level languages into machine readable format with added benefit of being interactive like an interpreter.
- Programming errors prevent the program from being compiled or executed.
- Syntax errors are words or symbols unrecognized by a particular programming language.
- Runtime errors only occur during program execution mostly due to an invalid input.
- Logical errors are considered when incorrect results are obtained based on provided input.
- Logical errors do not interrupt program execution.
- Integrated Development Environments (IDEs) are programs that facilitate writing, compiling and executing codes.
- IDEs usually provide a single environment for programmers to write and executes codes efficiently.
- C++ is a general-purpose high-level programming language.
- Reserved words are part of programming language syntax and cannot be used as name of variable, function or label.
- A constant is a named identifier having a value that cannot be changed.
- A variable is a named identifier with a value that can be changed during normal execution of program.
- Different types of values can be stored in variables. These types are called data types such as int, string, bool, etc.
- A variable can be declared by giving it a name and type. It can also be initialized during declaration by assigning a value to it.
- In C++, a variable is defined and initialized as:

```
"data_type variable_name= value;"
```

- C++ offers various data types for holding values in variables.
- These data types allocate system memory based on its type.

A. ENCIRCLE THE CORRECT ANSWER:

1. A computer program is a collection of:
   a. Tasks b. Instructions
   c. Computers d. Programmers
2. High-level languages have syntax that is:
   a. Easily readable by humans b. Easily readable by machines
   c. Easily readable by both d. None of the above
3. Low-level languages have syntax that is:
   a. Easily readable by humans b. Easily readable by machines
   c. Easily readable by both d. None of the above
4. The primary characteristic of a compiler is to:
   a. Translate codes line-by-line
   b. Translate low-level code to machine language
   c. Detect logical errors
   d. Translate codes all at once
5. The primary characteristic of an interpreter is to:
   a. Translate codes line-by-line
   b. Translate low-level code to machine language
   c. Detect logical errors
   d. Translate codes all at once
6. An Integrated Development Environment facilitates a programmer to:
   a. Edit source code b. Complete and highlight syntaxes
   c. Debug and compile codes d. All of the above
7. All errors, detected by users are typically:
   a. Syntax Errors b. Semantic Errors
   c. Run- Time Errors d. Logical Errors
8. Allowed names for declaring a variable:
   a. Can contain whitespaces
   b. Can be one of the reserved words
   c. Can contain letters, digits and underscores
   d. Can be the same as its data type

9. A bool data can store following type of value:
   a. Numbers b. Strings
   c. Fractional numbers c. True or false
10. Which data type occupies the most space in memory?
   a. Character b. Integer
   c. Floating point d. Double floating point

B. RESPOND THE FOLLOWING:

1. What is computer program?
2. List five common high-level languages used and describe their purpose.
3. Using the rules of naming variable, develop ten meaningful and valid variable names.
4. Write and two differences between machine and assembly language..
5. What are Strings in C++?
6. What is the difference between declaring and initializing a variable?
7. What is the difference between source code and object code?
8. List any four advantages of using an IDE.

1. In groups, students should learn to download, install and configure Dev C++.
2. Teacher demonstrates the use of IDE and its features as given in this unit. Also explains the use of variable and constants.