LAB # 1
To explain C-language programming concepts using Code::Blocks IDE

Objectives

- How To understand how to use Code::Blocks IDE
- To revise the basics of C-language programming

Pre-Lab

Building a Project using Code::Blocks IDE

1. Run Code::Blocks.
   Double click the icon
2. Let's start out with a very simple project.
   Select "File-> {} New-> {} Project…
   Select" Console Application "and click "Go".
   Move through the Wizard, choosing C++ as the language.
   Give the project the title "FirstProject", click on the "…" button to choose a place in which to save your work.
   Accept the rest of the defaults until the wizard closes.

3. On the left you can now see your project structure. If you open the Sources folder, you can see that it has already created a basic main.cpp file for you. Double-click this to open it in the editor.

4. Let's make some changes before compiling this. Change the function to look like this:

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

int main()
{
  string greeting = "Hello world!";
  cout <<
  return 0;
}
```

5. Save this and click the Build button (the yellow gear) to attempt compilation. You should see some compilation errors due to the incomplete statement.

6. Now put the cursor at the end of the "cout" line and type "greet" (without the quotes). Pause a moment and notice that a box pops up suggesting a possible completion for this variable name. Hit Tab to accept this suggestion. Now finish the program like this:

7. Save and again click the Build button. This time things should succeed.

8. Click the Run button (the blue triangle) to execute your program

Some important stuff

- Line 1. #include <iostream> is a header file library that lets us work with input and output objects, such as cout (used in line 5). Header files add functionality to C++ programs
- Line 2. using namespace std means that we can use names for objects and variables from the standard library.
- Line 3. A blank line. C++ ignores white space. But we use it to make the code more readable.
- Line 4. Another thing that always appear in a C++ program, is int main(). This is called a function. Any code inside its curly brackets {} will be executed.
- Line 5. cout (pronounced "see-out") is an object used together with the insertion operator (<<) to output/print text. In our example it will output "Hello World".
  Note: Every C++ statement ends with a semicolon ;
- Line 6. return 0 ends the main function.
- Line 7. Do not forget to add the closing curly bracket } to actually end the main function.

Omitting namespace

- Some C++ programs that runs without the standard namespace library. The using namespace std line can be omitted and replaced with the std keyword, followed by the :: operator for some objects:
  std::cout << "Hello World !";

C Variables and Constants

What is variable?

Variables in C have the same meaning as variables in algebra. A variable in C is a storage unit, which sets a space in memory to hold a value and can take different values at different times during program execution.

Rules to construct a valid variable name

i. A variable name may consists of letters, digits and the underscore (_) characters.
ii. A variable name must begin with a letter. Some system allows to starts the variable name with an underscore as the first character.
iii. ANSI standard recognizes a length of 31 characters for a variable name. However, the length should not be normally more than any combination of eight alphabets, digits, and underscores.
iv. Uppercase and lowercase are significant. That is the variable Totamt is not the same as totamt and TOTAMT.
v. The variable name may not be a C reserved word (keyword).

Some valid variable names

Total	Amount	ctr	name1
n1	M_age	AMOUNT	

Some invalid variable names

13th	(name)	111	%nm

Naming Conventions

Generally, C programmers maintain the following conventions for naming variables.

- Start a variable name with lowercase letters.
- Try to use meaningful identifiers
- Separate "words" within identifiers with mixed upper and lowercase (for example emp_Code) or underscores (for example emp_code).
- For symbolic constants use all uppercase letters (for example #define LENGTH 100, #define MRP 45).

Keywords and Identifiers

Every C word is classified as either a keyword or an identifier. Each keyword has a specific meaning, and these meanings cannot be changed. Keywords serve as basic building blocks for program statements. There are only 32 keywords in C. The list of all keywords in ANSI C is listed in the following table. All keywords must be written in lowercase.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

C Data Types

Data Types

Each program needs a certain kind of data for displaying a meaningful result. This certain kind of data are known as a data type.

ANSI C supports four classes of data types :

- Primary data types
- User-defined data types
- Derived data types
- Empty data set

All C compilers support four fundamental data types

Type	Range of values	Description
char (Characters)	-128 to 127	a single byte(8 bits) and can store one character type data

int Integers (whole numbers)	-32768 to 32767	an integer type is used to represent whole numbers within a specified range of values.
float Floating point (real numbers)	$3.4e^{-38}$ to $3.4e^{+38}$	single-precision floating point
double (Double)	$1.7e^{-308}$ to $1.7e^{+308}$	double-precision floating point

The following table shows the size and range of the type-specifiers on most common implementations:

Type	Size(bits)	Range
char or signed char	8	-128 to 127
unsigned char	8	0 to 255
int or signed int	16	-32768 to 32767
unsigned int	16	0 to 65535
short int or signed short int	8	-128 to 127
unsigned short int	8	0 to 255
long int or signed long int	32	2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
float	32	3.4E-38 TO 3.4E+38
double	64	1.7E-308 TO 1.7E+308
long double	80	3.4E-4932 TO 1.1E+4932

C Operators

C supports a wide variety of operators, such as +, -, *, /, &, <, > etc.,
Definition: An Operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables.
C operators can be classified into several categories. They are:

- Arithmetic operators.
- Relational Operators.
- Logical Operators.
- Assignment Operators.
- Increment and Decrement Operators.
- Conditional Operators.
- Bitwise Operators.
- Special Operators.

Arithmetic Operators:

Arithmetic operators include +, -, *, /, %, which performs all mathematical manipulations. These operators can operate on any built-in data type allowed in C. Table shows the function of Arithmetic Operators.

Operators	Meaning
+	Addition or unary plus
-	Subtraction of unary minus
*	Multiplication
/	Division
%	Modulo Division

Relational Operators:

Relational operators are used to compare two operands, and depending on their relation, certain decisions are taken. For example, it can be used to compare the age of two persons, price of two items and so on. There are 6 types of relational operators. They are:

Operators	Meaning
<	Is less than
>	Is greater than
<=	Is less than or equal to
>=	Is greater than or equal to
==	Is equal to
!=	Is not equal to

Logical Operators:

C supports three Logical Operators. They are:

Operators	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Assignment Operators

Assignment operators are used to assign the result of an expression to a variable. Usually, '=' operator is used. There is an additional 'shorthand' assignment operators of the form

V op = exp;

Here,

V= variable

exp = expression and

op = a binary arithmetic operator.

The Operator op= is known as the shorthand assignment operator.

Shorthand Assignment Operators

Statement with simple assignment operator	Statement with shorthand operator
a = a + 1	a += 1
a = a - 1	a -= 1
a = a * (n+1)	a *= n + 1
a = a / (n+1)	a /= n + 1
a = a % b	a % = b

Increment and Decrement Operators:

C supports two unique operators that are not found in other languages. They are: ++ and -- (increment and decrement operators respectively).

The operator ++ adds 1 to the operand, while – subtracts 1. Both are unary operators and take the following form:

++m; or m++;

--m; or m--;

++m is equivalent to m = m + 1;

--m is equivalent to m = m – 1;

Conditional Operator:

A ternary operator pair "? :" is available in C to construct conditional expressions of the form:

Exp1? Exp2: Exp3

Exp1, Exp2 and Exp3 are expressions. The operator ?: works as follows:

Exp1 is evaluated first. If it is nonzero (true), then exp2 expression is evaluated and becomes the value of the expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the expression. Here only one of the expressions is evaluated.

Bitwise Operators:

Bitwise operators are special operators that are used for manipulation of data at the bit level. These operators are used for testing the bits, or shifting them to right or left. Bitwise operators may not be applied to float or double.

Operators	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift Left
>>	Shift Right
~	One's Complement

In-Lab Tasks

Lab Task 1: Write a C++ program that takes 10 inputs from user, save it in a one-dimensional array and then find the sum of array elements and display the result

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____

LAB # 2
Explain the usage of pointers, functions and structures using C++ programming language

Objectives

- To explain the usage of pointers and its implementations in C++
- To explain the usage of functions and its implementations in C++
- To explain the usage of structures and its implementations in C++

Pre-Lab

Pointers

In C++, a pointer refers to a variable that holds the address of another variable. Like regular variables, pointers have a data type. For example, a pointer of type integer can hold the address of a variable of type integer. A pointer of character type can hold the address of a variable of character type.

Pre-Lab Task1: Printing Variable Addresses in C++

Write the given code in CodeBlocks IDE and understand the output

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

int main()
{
    // declare variables
    int var1 = 3;
    int var2 = 24;
    int var3 = 17;

    // print address of var1
    cout << "Address of var1: "<< &var1 << endl;

    // print address of var2
    cout << "Address of var2: " << &var2 << endl;

    // print address of var3
    cout << "Address of var3: " << &var3 << endl;
}
```

Get the Value from the Address Using Pointers

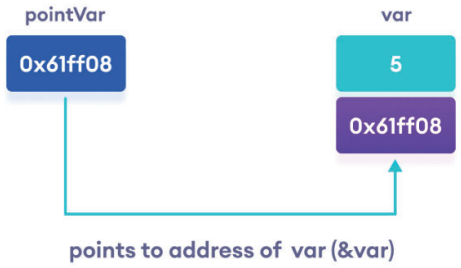
To get the value pointed by a pointer, we use the * operator. For example:

```
int* pointVar, var;
var = 5;

// assign address of var to pointVar
```

```
    pointVar = &var;

    // access value pointed by pointVar
    cout << *pointVar << endl;   // Output: 5
```

In the above code, the address of var is assigned to **pointVar**. We have used the ***pointVar** to get the value stored in that address.

When * is used with pointers, it's called the dereference operator. It operates on a pointer and gives the value pointed by the address stored in the pointer. That is, *pointVar = var.



Functions

A function is a block of code that performs a specific task.
Suppose we need to create a program to create a circle and color it. We can create two functions to solve this problem:

- a function to draw the circle
- a function to color the circle

Dividing a complex problem into smaller chunks makes our program easy to understand and reusable.
There are two types of function:

- Standard Library Functions: Predefined in C++
- User-defined Function: Created by users

C++ User-defined Function

C++ allows the programmer to define their own function.
A user-defined function groups code to perform a specific task and that group of code is given a name (identifier).
When the function is invoked from any part of the program, it all executes the codes defined in the body of the function.

C++ Function Declaration

The syntax to declare a function is:

```
returnType functionName (parameter1, parameter2,...) {
    // function body
}
```

Here's an example of a function declaration.

```
 // function declaration
void greet() {
    cout << "Hello World";
}
```

Here,

- the name of the function is greet()
- the return type of the function is void
- the empty parentheses mean it doesn't have any parameters
- the function body is written inside {}

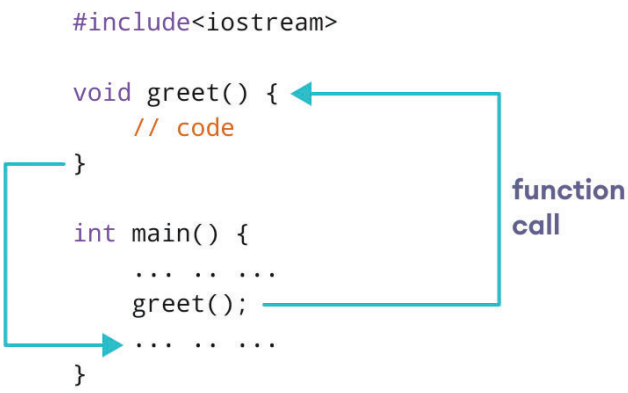
Note: We will learn about returnType and parameters later in this tutorial.

Calling a Function

In the above program, we have declared a function named greet(). To use the greet() function, we need to call it.

Here's how we can call the above greet() function.

```
int main() {
    // calling a function
    greet();
}
```



How Function works in C++

Pre-Lab Task2: Write the code and check the output

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

// declaring a function
void greet() {
    cout << "Hello there!";
}
int main() {
    // calling the function
    greet();
    return 0;
}
```

Function Parameters

As mentioned above, a function can be declared with parameters (arguments). A parameter is a value that is passed when declaring a function.

For example, let us consider the function below:

```
void printNum(int num) {
    cout << num;
```

```
}
```

Here, the int variable num is the function parameter.
We pass a value to the function parameter while calling the function.

```
int main() {
    int n = 7;

    // calling the function
    // n is passed to the function as argument
    printNum(n);

    return 0;
}
```

Pre-Lab Task3: Write the code and check the output

```
// program to print a text

#include <iostream>
using namespace std;

// display a number
void displayNum(int n1, float n2) {
    cout << "The int number is " << n1;
    cout << "The double number is " << n2;
}

int main() {

     int num1 = 5;
     double num2 = 5.5;

    // calling the function
    displayNum(num1, num2);

    return 0;
}
```

In the above program, we have used a function that has one int parameter and one double parameter. We then pass num1 and num2 as arguments. These values are stored by the function parameters n1 and n2 respectively.

C++ function with parameters

Note: The type of the arguments passed while calling the function must match with the corresponding parameters defined in the function declaration.

Return Statement

In the above programs, we have used void in the function declaration. For example,

```
void displayNumber() {
    // code
}
```

This means the function is not returning any value.
It's also possible to return a value from a function. For this, we need to specify the returnType of the function during function declaration.

Then, the return statement can be used to return a value from a function.
For example,

```
int add (int a, int b) {
   return (a + b);
}
```

Here, we have the data type int instead of void. This means that the function returns an int value.
The code return (a + b); returns the sum of the two parameters as the function value.
The return statement denotes that the function has ended. Any code after return inside the function is not executed.

Structures

Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collecion of data of different data types.

For example: You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables name, citNo, salary to store these information separately.

However, in the future, you would want to store information about multiple persons. Now, you'd need to create different variables for each information per person: *name1, citNo1, salary1, name2, citNo2, salary2*

We can easily visualize how big and messy the code would look. Also, since no relation between the variables (information) would exist, it's going to be a daunting task.

A better approach will be to have a collection of all related information under a single name Person, and use it for every person. Now, the code looks much cleaner, readable and efficient as well.

This collection of all related information under a single name Person is a structure.

How to declare a structure in C++ programming?

The ***struct*** keyword defines a structure type followed by an identifier (name of the structure).

Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct Person
{
    char name[50];
    int age;
    float salary;
};
```

Here a structure person is defined which has three members: name, age and salary.

When a structure is created, no memory is allocated.

The structure definition is only the blueprint for the creating of variables. You can imagine it as a datatype. When you define an integer as below:

int foo;

The int specifies that, variable foo can hold integer element only. Similarly, structure definition only specifies that, what property a structure variable holds when it is defined.

Note: Remember to end the declaration with a semicolon (;)

How to define a structure variable?

Once you declare a structure person as above. You can define a structure variable as:

```
Person bill;
```

Here, a structure variable bill is defined which is of type structure Person.

When structure variable is defined, only then the required memory is allocated by the compiler.

Considering you have either 32-bit or 64-bit system, the memory of float is 4 bytes, memory of int is 4 bytes and memory of char is 1 byte.

Hence, 58 bytes of memory is allocated for structure variable bill.

How to access members of a structure?

The members of structure variable is accessed using a dot (.) operator.

Suppose, you want to access age of structure variable bill and assign it 50 to it. You can perform this task by using following code below:

```
bill.age = 50;
```

Pre-Lab Task4: Write the code and check the output

C++ Program to assign data to members of a structure variable and display it.

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

struct Person
{
    char name[50];
    int age;
    float salary;
};

int main()
{
    Person p1;
```

```
        cout << "Enter Full name: ";
        cin.get(p1.name, 50);
        cout << "Enter age: ";
        cin >> p1.age;
        cout << "Enter salary: ";
        cin >> p1.salary;

        cout << "\nDisplaying Information." << endl;
        cout << "Name: " << p1.name << endl;
        cout <<"Age: " << p1.age << endl;
        cout << "Salary: " << p1.salary;

        return 0;
    }
```

In-Lab Tasks

Lab Task 1: Write a C++ program that takes 2 inputs from the user and return the result of it.

Use call by reference approach for function calling

Lab Task2: Write a C++ program that create a new data type called student with following attributes

1- reg_number
2- name
3- age
4- cgpa

Ask the user to enter 4 values in an array of student data type.

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____