NANYANG POLYTECHNIC
EGR204 Microcontroller Applications

E-Laboratory Session B

Course: Diploma in Robotics and Mechatronics

Module: EGR204 Microcontroller Applications

Experiment: E-Lab B

Title: Basic C programming II

Objective:

- Able to use logical operators, conditions and decision, and loop in the development of a C program

Learning Objectives:

- Understand logical operators.
- Understand condition and decisions.
- Understand loops.

1. Logical operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 (true) and variable **B** holds 0 (false). Notice the difference between logical operators and bit-wise operators. Evaluation of logical operator can only be true (1) or false (0), no other value s possible.

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

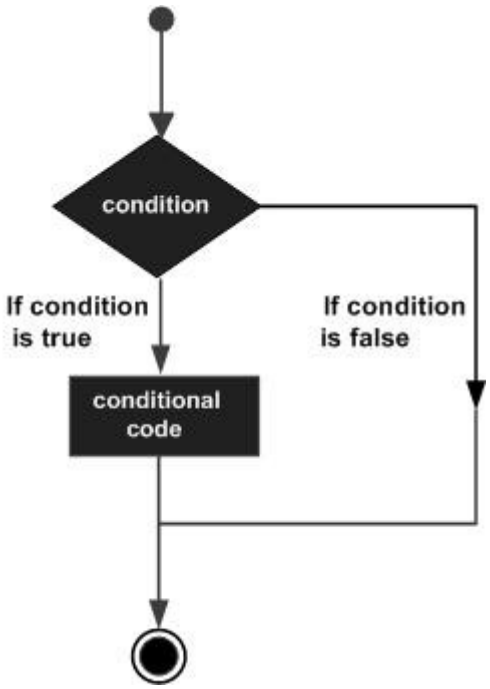
Type in the following code in repl editor and compare the difference between logical operator and bit-wise operator:

```
#include <stdio.h>

int main(void) {
  unsigned int A, B;
  A = 0b00000001;  // A = 1
  B = 0b00000010;  // B = 2
  printf("1) A&B = %d\n", A&B);
  printf("2) A|B = %d\n", A|B);
  printf("3) A&&B = %d\n", A&&B);
  printf("4) A||B = %d\n", A||B);
  return 0;
}
```

Output is:

2. Condition and decisions

Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.



C programming language provides the following types of decision making statements.

Sr.No.	Statement & Description
1	if statement An **if statement** consists of a boolean expression followed by one or more statements.
2	if...else statement An **if statement** can be followed by an optional **else statement**, which executes when the Boolean expression is false.
3	nested if statements You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).
4	switch statement A **switch** statement allows a variable to be tested for equality against a list of values.

5	nested switch statements You can use one **switch** statement inside another **switch** statement(s).

Type in the following code in repl editor and write down the output:

```
#include <stdio.h>
#define True 1
#define False 0
int main(void) {
  unsigned int A, B;
  unsigned int C;
  A = True;
  B = False;
  C = 20;

  // 1) if with one condition
  if(A == True)
  {
    printf("A is True\n");
  }
  // 2) if with multiple conditions
  if(A == True && B == False)
  {
    printf("A is True and B is False\n");
  }
  // 3) if else
  if (C<15)
  {
    printf("C is less than 15\n");
  }
  else
  {
    printf("C is greater than or equal to 15\n");
  }
  // 4) if else if
  if (A != True || B == True)
  {
    printf("A is False and B is True\n");
  }
  else if (A == True || B == True)
  {
    printf("Either A is True or B is True\n");
  }
  // 5) nested if else
  if(C>15)
  {
```

```
    if(C<25 && C>21)
    {
      printf("C is in (21, 25)\n");
    }
    else
    {
      printf("C is in (15, 21)\n");
    }
  }
  return 0;
}
```

Output is:

_____

_____

_____

_____

_____

_____

`Switch case`: A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

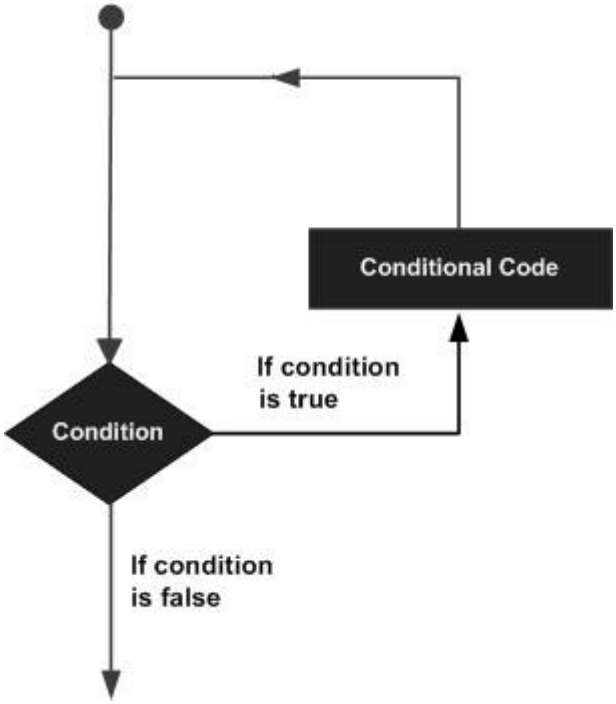
Type the following code in repl editor, change the value of A and observe the output. Modify the program so that you can have a case for A = 4.

```
#include <stdio.h>

int main(void) {
  unsigned int A=3;
  switch(A)
  {
   case 0:
      printf("A is 0\n");
      break;
   case 1:
      printf("A is 1\n");
      break;
    case 2:
      printf("A is 2\n");
    // you can have any number of cases
    default:
```

```
        printf("A is non of the above\n");
        break;
    }

    return 0;
}
```

3. Loops

You may encounter situations, when a block of code needs to be executed several numbers of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages −



C programming language provides the following types of loops to handle looping requirements.

Sr.No.	Loop Type & Description
1	while loop Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

2	for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	do...while loop It is more like a while statement, except that it tests the condition at the end of the loop body.
4	nested loops You can use one or more loops inside any other while, for, or do..while loop.

While loop: A while loop in C programming repeatedly executes a target statement as long as a given condition is true. Type the following code in repl editor and write down the output:

```
#include <stdio.h>

int main(void) {
  unsigned int A=0;
  while(A < 5)
  {
    printf("A is %d\n", A);
    A++;
  }
  return 0;
}
```

Output is:

_____

_____

_____

_____

_____

_____

For loop: A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. Type the following code in repl editor and write down the output:

```c
#include <stdio.h>

int main(void) {
  unsigned int A;
  for (A=0; A<5; A++)
  {
    printf("A is %d\n", A);
  }

  return 0;
}
```

Output is:

_____

_____

_____

_____

_____

_____

4. Assignment submission

Submission details: submit your source file as "E_Lab_B_assignment_(your admin number).c" at Blackboard Assignment "E-Lab B Assignment" Link.

```
#include <stdio.h>

int main(void) {

  _____________________          // declare a variable

  _____________________          // a for loop increments A from 0 to 100
  {
    _____________________        // check if A is equal to 99
    {
      printf("Last A is %d\n", A);
    }

    _____________________        // switch case with expression A
    {
      _____________________      // case for A = 10
        printf("A is %d\n",A);
        break;
      _____________________      // case for A = 20
        printf("A is %d\n",A);
        break;
      default:
        break;
    }
  }

  return 0;
}
```