# Laboratory Activity 6

## C- Conditional Statements: If and If-Else

### Objectives:

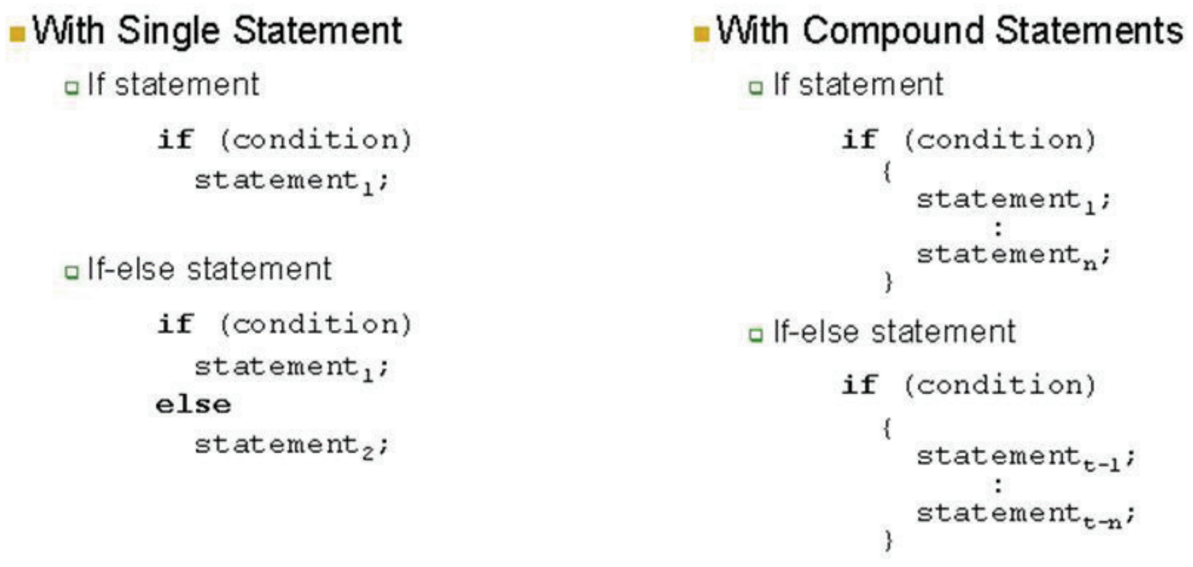
1. Learn the conditional statements of C-language using if and if-else.
2. Develop algorithms and flowcharts for use in programming applications.
3. Design, compile, test, run, and implement C language program

### Discussion:

### If and if-else statement

- ✓ A statement in C used to select one or two alternative course of actions.
- ✓ If statement is used in single alternative structure where it executes only when the condition is TRUE.
- ✓ If-else statement is used in two alternative structures, where it executes when the condition is either TRUE or FALSE.

### Syntax of if and if-else



- With Single Statement
  - If statement

```
if (condition)
  statement_1;
```

  - If-else statement

```
if (condition)
  statement_1;
else
  statement_2;
```

- With Compound Statements
  - If statement

```
if (condition)
  {
    statement_1;
        :
    statement_n;
  }
```

  - If-else statement

```
if (condition)
  {
    statement_t-1;
        :
    statement_t-n;
  }
```

**If-else statement**

```
if (condition) statement 1;
else
statement 2;
```

When the condition is proven true, statement 1 is executed. Otherwise, Statement 2 will run.

Else part is optional which means that it can be included or not.

## C- Logical Operators

&& - The logical-AND operator produces the value 1 if both operands have nonzero values. If either operand is equal to 0, the result is 0. If the first operand of a logical-AND operation is equal to 0, the second operand is not evaluated.

|| - The logical-OR operator performs an inclusive-OR operation on its operands. The result is 0 if both operands have 0 values. If either operand has a nonzero value, the result is 1. If the first operand of a logical-OR operation has a nonzero value, the second operand is not evaluated.

## Relationship and Equality Operators

| Operator | Relationship Tested |
|---|---|
| < | First operand less than second operand |
| > | First operand greater than second operand |
| <= | First operand less than or equal to second operand |
| >= | First operand greater than or equal to second operand |
| = = | First operand equal to second operand |
| != | First operand not equal to second operand |

Example 1: Write a program that determines if the input age is qualified to vote or not. The qualifying age is 18 years old and above.

```
Algorithm:
Enter age
If age > 18 : “Qualified”
Else: “Too young!”
```

```
#include   <stdio.h>
#include  <stdlib.h>
int main()
{
  int age;
  printf("\nEnter age: ");
  scanf("%d", &age);
  if (age>=18)
    printf("Qualified to vote\n");

  else
     printf("Too Young!\n");

  system("PAUSE");
  return 0;
}


//using logical operators
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int age;
  printf("\nEnter age: ");
  scanf("%d", &age);

  if ((age>=1)&&(age<=100))
      {
      if (age>=18)
       printf("qualified to vote\n");
       else
       printf("too young!");
       }
 else
        printf("out of range!\n");

  system("PAUSE");
  return 0;
}
```

**Example 2:** Write a program that accepts the input magic number. If the input is right, the magic words will be displayed. The magic number is 143 and its corresponding magic words are: "I love you!" if the input number is wrong, displays: "Sorry, better luck next time".

```
#include   <stdio.h>
#include  <stdlib.h>
int main()
{
    int mn;
    printf("Enter a magic number: ");
    scanf("%d", &mn);
    if (mn==143)
       printf("I love you!\n");
```

```
   else
    printf("Sorry, better luck next time!\n");

 system("PAUSE");
 return 0;
}
```

**Example 3:** Write a program that determines if the input number is POSITIVE or NEGATIVE. Consider 0 as positive, considering that it contains no negative sign.

```
#include   <stdio.h>
#include  <stdlib.h>
int main()
{
   int n;
   printf("\nEnter a number:");
   scanf("%d", &n);

   if(n>=0)
         printf("Positive\n");
   else
         printf("Negative\n");

 system("PAUSE");
 return 0;
}
```

**Example 4:** Write a program that determines if the input number is ODD or EVEN number. Use %= modulo

The modulo operation finds the remainder of division of one number by another.

For instance, the expression "5 mod 4" would evaluate to 1 because 5 divided by 4 leaves a remainder of 1, while "9 mod 3" would evaluate to 0

```
#include   <stdio.h>
#include  <stdlib.h>
int main()
{
   int r,n;
   printf("\nEnter a number:");
   scanf("%d", &n);
   r=n % 2;
   if(r==0)
         printf("EVEN\n");
   else
         printf("ODD\n");
  system("PAUSE");
  return 0;
}
```

*"The sequence of if-else statements is the most general way of writing a multi-way decision. The expressions are evaluated in order, if any expression is true, the statement associated with it is executed, and this terminates the whole chain."*

*-Dennis Ritchie*

# Laboratory Activity 7

## C- Conditional Statements: Ladderized If-Else and Switch/Case

### Objectives:

1. Learn the conditional statements of C-language using ladderized if-else and case/switch statements.
2. Develop algorithms and flowcharts for use in programming applications.
3. Design, compile, test, run, and implement C language program

### Discussion:

### Ladderized if-else

Only those conditional statements and expressions that were first proven true are the ones to be chosen and its associated statements will be executed. When all conditions were proven false, else statement x will run.

Syntax:

```
if (condition 1)
        statement 1;
  else if (condition )
        statement 2;
  else if (condition )
        statement n;
else
        statement x;
```

**Example 1:** Write a program to assist a teacher in calculating student grades at the end of the semester. It accepts a numerical grade as input, then it will display the character as output, based on the given scale:

| | |
|---|---|
| 90 and above: | A |
| 80-89 | B |
| 70-79 | C |
| 60-69 | D |
| Below 60 | F |

```
#include   <stdio.h>
#include  <stdlib.h>
int main()
{
   int g;
   printf("\nEnter grade: ");
   scanf("%d", &g);
   if ((g>=90) && (g<=100))
       printf("A\n");

   else if ((g>=80) && (g<=89))
       printf("B\n");
   else if ((g>=70) && (g<=79))
       printf("C\n");
    else if ((g>=60) && (g<=69))
       printf("D\n");
    else if ((g>=50) && (g<=59))
       printf("F\n");

    else
       printf("Out of Range\n");

  system("PAUSE");
  return 0;
}
```

**Example 2:** Write a program that displays an equivalent color once an input letter match its first character, for example b for Blue, r for Red, and so on. Here are the given criteria:

| | |
|---|---|
| 'B' or 'b' | Blue |
| 'R' or 'r' | Red |
| 'G' or 'g' | Green |
| 'Y' or 'y' | Yellow |

```
#include   <stdio.h>
#include  <stdlib.h>
int main()
{
  char let;
  printf("Enter a letter: \n");
  scanf("%c", &let);

  if((let=='B') || (let=='b'))
  printf("Blue\n");

  else if((let=='R') || (let=='r'))
  printf("Red\n");
  else if((let=='G') || (let=='g'))
  printf("Green\n");

  else if((let=='Y') || (let=='y'))
  printf("Yellow\n");

  else
  printf("Unknown Color\n");
 system("PAUSE");
 return 0;
}
```

## Switch/Case Statement

Used to simplify some of the tasks of ladderized if-else statements. Variables declared as integers and characters are the best candidates that can be tested or evaluated by the switch.

*"The switch/case statement is a multi-way decision that tests whether an expression matches one of a number of constant integer values, and branches accordingly. If a case matches the expression value, execution starts at the case."* -Dennis Ritchie

```
switch(var_expression) {
case const_value: statement 1; break;
case const_value: statement 2; break;
case const_value: statement 3; break;
case const_value: statement 4; break;
default: statement x; break;
}
```

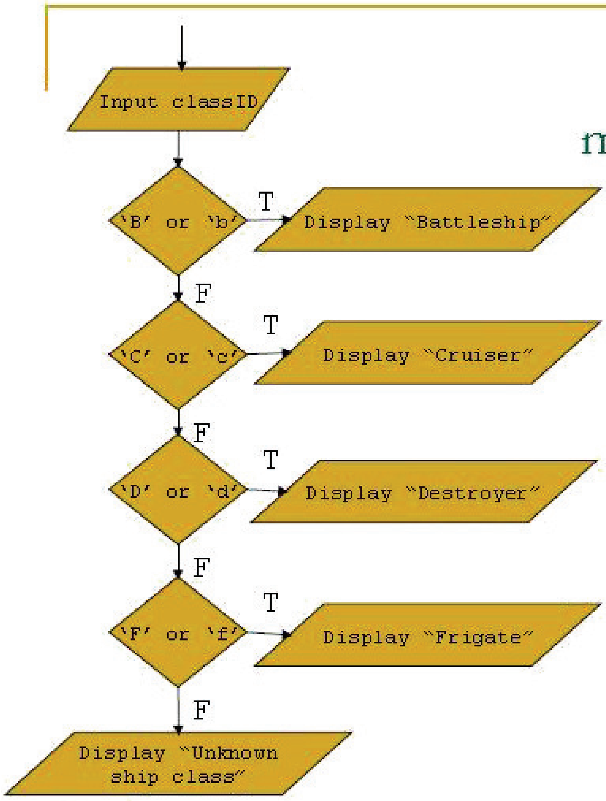
*"The break statement causes an immediate exit from the switch. Because cases serve just as labels, after the code for one case is done, execution falls through to the next unless you take explicit action to escape."* -Dennis Ritchie

**Example 3:** Write a program to display the high school level of a student, based on its year-entry number.

| | |
|---|---|
| 1 | Freshman |
| 2 | Sophomore |
| 3 | Junior |
| 4 | Senior |

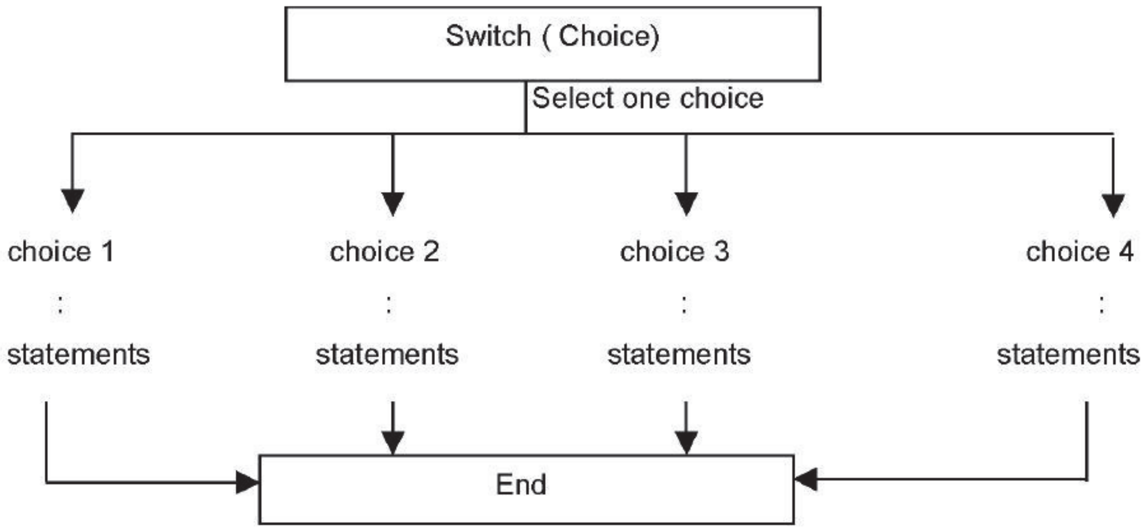
```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  int n;
  printf("enter your year-entry number: \n");
  scanf("%d",&n);

  switch(n)
  {
  case 1: printf("Freshman\n"); break;
  case 2: printf("Sophomore\n"); break;
  case 3: printf("Junior\n"); break;
  case 4: printf("Senior\n"); break;
  default: printf("Out-of-School\n"); break;
  }
  system("PAUSE");
  return 0;

}
```

## *Switch* statement in multiple selection structure



```
switch (class) {
  case 'B':
  case 'b':
            printf("Battleship \n");
  case 'C':
  case 'c':
            printf("Cruiser \n");
  case 'D':
  case 'd':
            printf("Destroyer\n");
  case 'F':
  case 'f':
            printf("Frigate\n");
  default:
            printf("Unknown ship class
\n");                }
```

### Case/Switch Structure



# Laboratory Activity 8

# C- Looping Statements (Part 1)

## Objectives:

1. Learn the looping statements of C-language such as for, while and do-while loop.
2. Develop algorithms and flowcharts for use in programming applications.
3. Design, compile, test, run, and implement C language program

## Discussion:

## Looping Statement

A program instruction that repeats some statement or sequence of statements in a specified number of times.

A set of instructions to be performed all over again until a certain condition is reached, met, or proven and tested as false.

*"Loops are used to repeat a block of code. Being able to have your program repeatedly execute a block of code is one of the most basic but useful tasks in programming -- many programs or websites that produce extremely complex output (such as a message board) are really only executing a single task many times."* –Alex Alain, cprogramming.com

## Three Looping Statements of C-Language

### 1. For-loop

The for loop is the most powerful and flexible of all the loops used in C. the general syntax is:

```
for(initialization ; condition ; loop control)
{
        …action;
        …action; //loop
        body …action;
}
```

The initialization is evaluated before the loop begins. You have to declare and assign an initial value for the loop. This initialization is evaluated only once at the beginning of the loop.