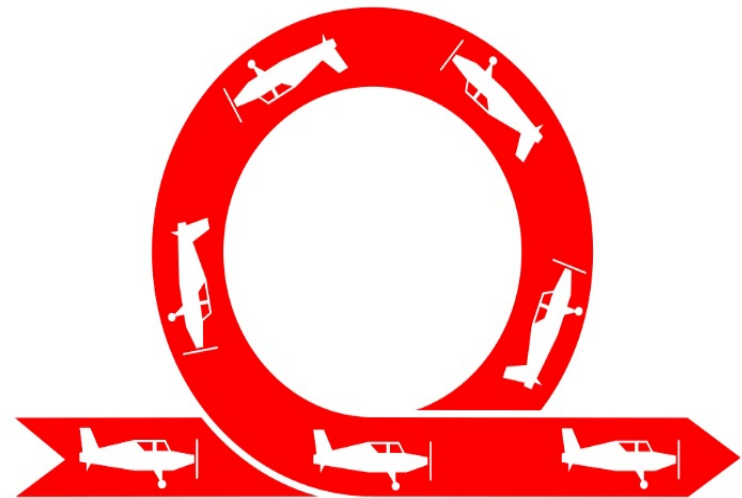# C Fundamental

Decision & Looping

Objectives

- Explain the Selection Construct
  - ✓ If Statement
  - ✓ If – else statement
  - ✓ Multi if statement
  - ✓ Nested if statement

- Switch statement
- Looping

Conditional Statement

- **Conditional statements enable us to change the flow of the program**

- **A conditional statement evaluates to either a true or a false value**

Example :

To find whether a number is even or odd we proceed as follows :

1. Accept a number

2. Find the remainder by dividing the number by 2

3. If the remainder is zero, the number is "EVEN"

4. Or if the remainder is not zero the number is "ODD"

C supports two types of selection statements

The if statement

The switch statement

The **if** statement-1

Syntax:

```
if (expression)
{

    statements;

}
```

If the **if** expression evaluates to true, the block following the **if** statement or statements are executed

The **if** statement-2

Program to display the values based on a condition

```c
#include <stdio.h>
void main()
{
  int x, y;
  char a = 'y';
  x = y = 0;
  if (a == 'y'){
    x += 5;
    printf("The numbers are %d and \t%d", x, y);
  }
}
```

Syntax:

```
if (expression)
{
    statements;
}
else
{
    statements;
}
```

Program to display whether a number is Even or Odd

```c
#include <stdio.h>
void main()
{
   int num , res ;
   printf("Enter a number :");
   scanf("%d",&num);
   res = num % 2;
   if (res == 0)
     printf("Then number is Even");
   else
     printf("The number is Odd");
}
```

The if–else–if statement-1

Syntax:

```
if (expressions) {
    statements;
}
else if(expressions) {
    statements;
}
else if(expressions) {
    statements;
}

    .
    .
    .
else {
    statements;
}
```

- The if – else – if statement is also known as the if-else-if ladder or the if-else-if staircase.

- The conditions are evaluated from the top downwards.

The if–else–if statement-2

Program to display a message based on a value

```c
#include <stdio.h>
void main()
{
    int x;
    x = 0;
    clrscr ();
    printf("Enter Choice (1 - 3) : ");
    scanf("%d", &x);
    if (x == 1)
        printf ("\nChoice is 1");
    else if ( x == 2)
        printf ("\nChoice is 2");
    else if ( x == 3)
        printf ("\nChoice is 3");
    else
        printf ("\nInvalid Choice ");
}
```

Nested if-1

Syntax:

```
if (expl)
{
    if (exp2) statement1;
    if (exp3) statement2;
    else statement3;          /*with if (exp3) */
}
else statement4;              /* with if (expl) */
```

- Note that the inner else is associated with **if(exp3)**

- According to ANSI standards, a compiler should support at least 15 levels of nesting

Nested if-2

```c
#include <stdio.h>
void main (){
    int x, y;
    x = y = 0;
    clrscr ();
    printf ("Enter Choice (1 - 3) : ");
    scanf ("%d", &x);
    if (x == 1){
        printf("\nEnter value for y (1 - 5) : ");
        scanf ("%d", &y);
        if (y <= 5)
            printf("\nThe value for y is : %d", y);
        else
            printf("\nThe value of y exceeds 5 ");
    }
    else
        printf ("\nChoice entered was not 1");
}
```

The switch statement-1

```
switch (expression)
{
        case constant1:
                statement sequence
                break;
        case constant2:
                statement sequence
                break;
        case constant3:
                statement sequence
                break;

        .

        .

        .

        default:
                statement sequence
}
```

Program to check whether the entered lowercase character is vowel or 'z' or a consonant

```c
#include <stdio.h>
  main ()
  {
     char ch;
     clrscr ();

     printf ("\nEnter a lower cased alphabet (a - z) : ");
     scanf("%c", &ch);
```

continued…….

```
if (ch < 'a' || ch > 'z')
    printf("\nCharacter not a lower cased alphabet");
else
    switch (ch) {
        case 'a' :
        case 'e' :
        case 'i' :
        case 'o' :
        case 'u' :
            printf("\nCharacter is a vowel");
            break;
        case 'z' :
            printf ("\nLast Alphabet (z) was entered");
            break;
        default :
            printf("\nCharacter is a consonant");
            break;
        }
    }
```

What is a Loop?

Section of code in a program
which is executed repeatedly,
until a specific condition is satisfied

3 types of Loop Structures

The for loop

The while loop

The do….while loop

The for loop-1

```
for (initialize counter; conditional test; re-evaluation parameter)
{
    statement
}
```

- The initialize counter is an assignment statement that sets the loop control variable, before entering the loop

- The conditional test is a relational expression, which determines, when the loop will exit

- The evaluation parameter defines how the loop control variable changes, each time the loop is executed

The for loop-2

```c
/*This program demonstrates the for loop in a C program */
#include <stdio.h>

void main()
{
    int count;
    printf("\tThis is a \n");

    for(count = 1; count <=6 ; count++)
        printf("\n\t\t nice");

    printf("\n\t\t world. \n");
}
```

The Comma Operator

The scope of the **for** loop can be extended by including more than one initializations or increment expressions in the for loop specification
The format is : **exprn1 , exprn2 ;**

```c
#include <stdio.h>
void main()
{
    int i, j , max;
    printf("Please enter the maximum value \n");
    printf("for which a table can be printed: ");
    scanf("%d", &max);

    for(i = 0 , j = max ; i <=max ; i++, j--)
        printf("\n%d  +  %d  =  %d",i, j, i + j);
}
```

- The **for** loop will be termed as a **nested for** loop when it is written as follows

```
for(i = 1; i<max1; i++)
    {
                .
                .
        for(j = 0; j < = max2; j++)
        {
            .
            .
        }
        .
        .
    }
```

Nested for Loops-2

```c
#include <stdio.h>
  void main()
  {
   int i, j, k;
   i = 0;
   printf("Enter no. of rows :");
   scanf("%d", &i);
   printf("\n");
   for (j = 0; j < i ; j++)
   {
      printf("\n");
      for (k = 0; k <= j; k++) /*inner for loop*/
      printf("*");
   }
  }
```

while (condition is true)
statement ;

The while loop repeats statements while a certain specified condition is True

The while Loop-2

```c
/* A simple program using the while loop */

#include <stdio.h>
void main()
{
  int count = 1;
  while( count <= 10)
  {
      printf("\n This is iteration %d\n",count);
      count++;
  }

  printf("\n The loop is completed. \n");
}
```

do...while Loop-1

```
do{
        statements;
} while (conditions);
```

- In the **do while** loop the body of the code is executed once before the test is performed

- When the condition becomes False in a **do while** the loop will be terminated, and the control goes to the statement that appears immediately after the **while** statement

```c
#include <stdio.h>
void main ()
{
  int num1, num2;
        num2 = 0;
  do
  {
     printf( "\nEnter a number : ");
     scanf("%d",&num1);
     printf( " No. is %d",num1);
     num2++;
  } while (num1 != 0);
  printf ("\nThe total numbers entered were %d",--num2);

 /*num2 is decremented before printing because count for last
 integer (0) is not to be considered */
}
```

Jump Statements-1

return expression

- The return statement is used to return from a function

- It causes execution to return to the point at which the call to the function was made

- The return statement can have a value with it, which it returns to the program

Jump Statements-2

goto   label

- The goto statement transfers control to any other statement within the same function in a C program

- It actually violates the rules of a strictly structured programming language

- They reduce program reliability and make program difficult to maintain

Jump Statements-2

break statement

- The break statement is used to terminate a case in a switch statement

- It can also be used for abrupt termination of a loop

- When the break statement is encountered in a loop, the loop is terminated immediately and control is passed to the statement following the loop

continue statement

- The continue statement causes the next iteration of the enclosing loop to begin

- When this statement is encountered, the remaining statements in the body of the loop are skipped and the control is passed on to the re-initialization step

Break statement

```c
#include <stdio.h>
void main ()
{
  int count1, count2;
  for(count1 = 1, count2 = 0;count1 <=100;  count1++)
  {
    printf("Enter %d count2 : ",count1);
    scanf("%d", &count2);
    if(count2==100) break;
  }
}
```

continue statement

```c
#include <stdio.h>
void main ()
{
    int num;
    for(num = 1; num <=100; num++)
    {
        if(num % 9 == 0)
            continue;
        printf("%d\t",num);
    }
}
```

Exit Function

exit() function

- The exit() is used to break out of the program

- The use of this function causes immediate termination of the program and control rests in the hands of the operating system

Thank you
Q&A