

## **Chapter 2: Decision Making, Loops and Control**

In this chapter you will learn all about:

- Decision-making
- Control loops
- Random number generation
- Other useful C library functions

=====

.

## A. Decision Making statements

We will see three different types of decision making statements – If / If-else / if-else if - else if-...else if - else / switch; Choice of writing depends on the context, ease of expression writing, modularity, etc. Underlying logic can be expressed in all possible types interchangeably.

### A.0 if-else Statement

#### General form

```
if (expression) {  
    < statements >  
}  
else {  
    < statements >  
}
```

#### Note:

-else part is optional

- in nested sequence, else is associated with previous if without an else

#### Example 2.1: [DIY] Test and run the following code

```
int x, y=10;  
x = 15;  
if(x <= y)  
    printf("Value of x[from if] is: %d\n", x);  
else {  
    printf("Value of x [from else] is: %d\n", x);  
    printf("Try another value of x");  
}
```

## A.1 if – else if - else

### Else-If is used for multiple decisions

#### General Form

```
if (expression)
    <statements>
else if (expression)
    <statements>
else if (expression)
    <statements>
else
    <statements>
```

### Expressions are evaluated in order

#### NOTE - When any expression is true

- the corresponding statements are executed, and the whole chain terminates

#### Example 2.2: Test and run the following code

```
x = 100; /* integer x */
if((x >= 1) && (x <= 10))
    printf("Value of x[from if] is: %d\n", x);
else if((x > 10) && (x <= 15)) {
    printf("Value of x [from elseif] is: %d\n", x);
    printf("You can try for other values of x");
}
else /* default case */
    printf("From else: default value of x: %d :", x);
```

## A.2. Case selection (switch)


Switch is a special multiple decision selector; Tests for constant expressions

### General form

```
switch (expression) {  
    case <constant expression>:  
        <statements>  
        break;  
    case <constant expression>:  
        <statements>  
        break;  
    case <constant expression>:  
        <statements>  
        break;  
    default: /* equivalent to your else construct */  
        <statements>  
}
```

**Alternate forms:** When multiple case labels are required, we have:

```
switch (expression) {  
    case <constant expression1>:  
    case <constant expression2>:  
    ...  
    case <constant expression k>:  
        <statements>  
        break;  
    default:  
        <statements>  
}
```



All these are possible values for executing the statements following the cases

***Case serves as a goto label for switch;*** Use **break** to prevent falling through other cases. [*Do NOT use goto statement in C.*]

**Example 2.3:** Interpret the following code.

```
char grade = 'B';
switch(grade) {
    case 'A' :
        printf("Excellent!\n" );
        break;
    case 'B' :
    case 'C' :
        printf("Well done\n" );
        break;
    case 'D' :
        printf("You passed\n" );
        break;
    case 'F' :
        printf("Better try again\n" );
        break;
    default :
        printf("You entered an invalid grade\n");
}
printf("Your grade is  %c\n", grade );
```

**Tutorial 2.1** – Implement a C code which can verify if a character entered is a consonant or a vowel using a switch() statement.

## Other uses of switch()

Suppose we want to check if a value belongs to an interval. We can use switch statement as follows:

**Example 2.4:** Switch statement using range of values

```
switch(number)
{
    /* case values within a range */
    case -25 ... 0: /* Observe the 3 dots; no gaps;- syntax */
        printf("Number is in between -25 to 0\n");
        break;
    /* case values within a range */
    case 1 ... 25:
        printf("Number is in between 1 to 25\n");
        break;
    /* default case */
    default:
        printf("Number is out of range!!!\n");
        break;
}
```

## B. Control Loops

### B.0 *while* loop

#### General form

```
while (expression) {  
    <statements>  
}
```

#### Example 2.5: Test and run your code

##### **Tutorial 2.2:** [Discuss] *What is the output from this piece of code?*

```
int x;  
x=7;  
while(x >=0) {  
    printf("%d\n",x);  
    x=x-2;  
}
```

**Q:** What if the two lines after while statement are swapped?

**Q:** What does **while(1)** condition means?

### B.1 *do – while* loop

#### General form

```
do {  
    <statements>  
} while (expression);
```

**Tutorial 2.3:** [Discuss] *What does this piece of code do?*

```
int num = 0;
do {
    printf("%d\n", num++);
} while(num <= 9);
```

**Note - Termination condition at the bottom of loop**

**Think!** *What is special about “while” & “do-while” statements?*

## B.2 *for* loop

**General form:**

```
for (expr1; expr2; expr3) {
    <statements>
}
```

**Note:** *expr1* will be usually the **start value**; *expr2* will be a **condition check**; *expr 3* will be an **auto inc/decrement** expression

### **Example 2.6:** Test and run your code

```
for (i = 7; i < 10; i++)
/* start value = 7; condition check → i < 10; auto increment*/
    printf("count = %d\n", i);
```

**Note:** You can also use like: **for**(i='A'; i<='Z'; i++) { ... } Define char i; and use.



**Tutorial 2.4** – *What will be printed?*

```
for(i='A'; i<='F'; i++) {  
    printf("%d, %c\n",i,i);  
    i +=2;  
}
```

## C. Random Number Generation

Random numbers can be generated in a variety of ways. In C, we use library function **rand()** to generate random (pseudo) numbers.

Consider the following code, which demonstrates a simple way to generate random numbers.

**Example 2.7: Run and test!**

```
#include <stdio.h>  
#include <stdlib.h>  
main() {  
    int c, mynum;  
    srand(time(NULL));  
    printf("20 random numbers in [1,35]\n");  
    for (c = 1; c <= 20; c++) {  
        mynum = rand()%35 + 1; /* why this? */  
        printf("%d\n", mynum);  
    }  
    return 0;  
}
```

### *How does the **rand()** work?*

Behind the **rand()** function, there exist a seed value which serves as a basis for every random number generated. First we use the **srand()** function to set the seed. Basically, the computer can generate random numbers based on the number that is fed to **srand()**. If you use the same seed value, then the same set of random numbers would be generated every time.

Therefore, we have to set the seed with different values, probably in every iteration. *How do we do this?*

We use the time function to feed the current time value to the srand() function which sets the seed for that iteration. Since time is a varying parameter, in every call to srand(), seed value will be different and random numbers will be different.

Of course, we can set one seed for every set of iterations (as in Example 4.11) and, not in every iteration, if we wish to. Thus the above code can be modified as follows.

**Tutorial 2.5: [Discuss]** Modify Exercise 2.7 to set the seed in every iteration.

### ***Bonus Problems [DIY]:***

- (a) Use **WHILE** loop to generate Fibonacci numbers for a given  $n \geq 1$ ;
- (b) Repeat (a) using **DO-WHILE()** loop.
- (c) What does the following code do?

```
for (i=0; i<=100; i++) {  
    printf("%d\n",i);  
    i++;  
    printf("%d\n",i);  
}
```

## C. Some useful C library functions\*\*

<b>&lt;stdio.h&gt;</b>	Nearly 50 functions available; All input and output (scanf, printf) functions, All file operations related operations; Contains some important macros like EOF (marks the end of file, which can be directly used in our programs)
<b>&lt;string.h&gt;</b>	String related operations – strcmp(...), strcpy(...), strrev(...), strlen(...), other pointer-based string functions, etc;
<b>&lt;time.h&gt;</b>	Time related functions – local time, calendar time(specific format), clock() function returns a value to a variable clock_t, etc;
<b>&lt;math.h&gt;</b>	All math functions – trigonometric, algebraic – sqrt(..), pow(...), log10(), log(), etc. Almost all input variables are of type double;

\*\* : Not an exhaustive list; You may Google “C library functions” to check the full list;

### Example 2.8: Determine the **running time** of your program!

This is an example of using **<time.h>** library. In order to determine the run time elapsed, you need to use the **clock()** function available in the **time.h** library.

**clock\_t clock()** function returns the number of clock ticks elapsed since the program was launched. On failure, the function returns a value of -1.

To get the number of seconds (actual time) used by the CPU, you need to divide by CLOCKS\_PER\_SEC. *How do I know this?*

On a 32 bit system where CLOCKS\_PER\_SEC equals 1000000 this function will return the same value approximately every 72 minutes. This is a standard pre-defined by the library function.

See the complete code below. You can use this logic to quantify the running time of your program!

```
#include <time.h> /* use this header file*/
#include <stdio.h>

int main()
{
    clock_t start_t, end_t, total_t; /* clock_t is a pre-defined structure */
    int i;
    printf("CLOCKS_PER_SEC = %ld\n",CLOCKS_PER_SEC);
    start_t = clock(); /* observe this */
    printf("Prog start time, start_t = %ld\n", start_t);
    printf("Going to scan a big loop, start_t = %ld\n", start_t);
    for(i=0; i< 100000; i++)
    {
    }
    end_t = clock(); /* observe this */
    printf("Big loop ends at time, end_t = %ld\n", end_t);

    /* now we need to calculate the actual time */

    total_t = (end_t - start_t) / CLOCKS_PER_SEC;
    printf("Total time taken by CPU: %ld\n", total_t );
    printf("End of this proram!\n");
    return(0);
}
```

## D. Implementing your logic involving loops & control

**Tutorial 2.6:** Implement the following problems.

- (a) Trace the following program and determine its output.

```
#include<stdio.h>

int main()
{
    int i,j,k;

    for(i=0 ; i<=5 ; i++)
    {
        for(j=5 ; j>i ; j--)
            printf(" ");

        for(k=0 ; k<=i; k++)
        {
            if(k%2==0)
                printf("A");
            else
                printf("B");
        }
        printf("\n");
    }

    return 0;
}
```

- (b) [**DIY**] Implement a code that adds all the numbers until user enters a zero!
- (c) Design a calculator program to carry out basic arithmetic operations for two operands using **switch()** statement.
- (d) Implement a C code using **while** loop to print the following sequence as shown.

```
(0,0) (0,1) (0,2) . . . (0,9)
(1,0) (1,1) (1,2) . . . (1,9)
.
.
.
(9,0) (9,1) (9,2) . . . (9,9)
```

(e) This has two parts:

- (i) Generate an arbitrary DNA sequence comprising **N** nucleotides from the set {A,G,C,T} as follows using a **switch** and a **control loop** of your choice:

Generated output:

AGCTAGGTACAGGATTAAA . . . TTCGTACCA

- (ii) [**DIY**] Set  $N = 100000$ . Measure the time taken for generating the output sequence.