Tutorial 3 : 16 August

1 If - Else if - Else ladder

Consider the following code and try to find the output.

```c
#include <stdio.h>

int main() {
    printf("What did the buffalo say to his son when he left for college?\n");
    int x = 70;
    if (x > 90) {
        printf("GoSon");
    } else if (x > 80) {
        printf("YoSon");
    } else if (x > 65) {
        printf("BiSon");
    } else if (x > 45) {
        printf("NoSon");
    } else{
        printf("SoSon");
    }
    return 0;
}
```

Output :
What did the buffalo say to his son when he left for college?
BiSon

Explanation :
The if-else ladder is computed in a top-down fashion till one of the conditions is satisfied. Then the statements corresponding to that topmost true condition is executed and the rest of the ladder is skipped. The else condition is run only when all the if and else if conditions are false.
So in this example, the topmost true condition is $(x > 65)$?. Thus we get the output *BiSon*.

2 Switch vs If-Else

- Switch can only be used for comparison based tests whereas if-else can be used for more complex conditions.
- Switch can be used for comparison based on one expression only whereas if-else can be used for multiple conditions.
- Adding and removing labels is much cleaner in switch than in if-else. Thus the code becomes more maintainable.

Generally if you have comparison based tests with more than 3 possible cases, switch would be preferred over if-else.

3 'break' and 'default' in switch

Consider the following code and try to find the output.

```
#include <stdio.h>

int main() {
    int i;
    for (i = 2; i < 16; i*=2) {
        //i*=2 is the same as i=i*2
        switch (i) {
            default :
                printf("no ");
            case 16 :
                printf("name.");
                break;
            case 4 :
                printf("boy ");
                break;
            case 6:
                printf("girl ");
                i--;
            case 12:
                printf("has ");
                i--;
                break;
            case 2:
                printf("A ");
                i++;
                break;
        }
    }
    return 0;
}
```

Output :
A girl has no name.

Explanation :
Let us consider the entire code in terms of iterations for different values of i.

- **Iteration 1, i = 2**
  We get output *A* and i becomes 3
- **Iteration 2, i = 6**
  Since we perform *i *= 2;* each time before beginning the next iteration, thus value of i becomes 6 in the second iteration. Note that there is no break after *case 6*, thus the control falls through to the next case and we get output *girl has*. Also by the end of second iteration, value of i becomes 4.
- **Iteration 3, i = 8**
  Since no case matches for $i = 8$, we execute the default case. Note that it is not necessary to put the default case at the end but is generally preferred. Again because of no break statement at the end of default, the control falls through to *case 16*.
  Overall, till now we get the output *A girl has no name.*
- **Iteration 4, i = 16**
  Finally the value of i becomes 16 and thus the condition of *for* loop is not satisfied. Thus we reach the return statement and the program terminates.

4 Revisiting scanf

It is important to understand that all input specifiers **other than** %c, %n and %[] ignore all the **leading** whitespaces before the input specifier. Whitespaces mentioned here include spaces(' '), tabs('\t') and newline('\n') characters. The folowing code examples will give a better picture.

```
#include <stdio.h>

int main(void) {
    int x;
    scanf("%d", &x);
    printf("Number entered: %d", x);
    return 0;
}
```

Input:

  45

Output:
Number entered: 45

Explanation:
If not clearly visible, the input string is of the form:
(Remember: The input and output are always in the form of characters)

'\n'	' '	' '	'4'	'5'

As can be seen, whenever the user presses *ENTER* in the input, it is recorded as a '\n' by the computer. What %d as input specifier does is ignore all the leading whitespaces (whitespaces before the first number is encountered) and directly starts reading from the position of '4'. Thus we get the above mentioned output.
To understand the behaviour of %c, consider the following code example.

```
#include <stdio.h>

int main ()
{
  char a;
  scanf ("%c",&a);
  int x = a;
  printf ("Character entered has ASCII code : %d\n", x);
  return 0;
}
```

Input:

```


  a
```

Output:
Character entered has ASCII code : 10

Explanation: If not clearly visible, the input string is of the form:

'\n'	'\n'	' '	' '	'a'

Thus what %c as input specifier does is directly read the first character in the input buffer(which is '\n') and the code displays its ASCII code in the output(which is 10).

A common workaround is to provide a whitespace character before %c in the format string, as shown:

```
scanf (" %c",&a);
```

This works because for even a single whitespace character in format string (here, the space before %c), the scanf function ignores any number of whitespace characters(0 or more) encountered before the next non-whitespace character.

Another workaround used sometimes is the function *getchar()*. This function reads and returns a single character from input. The following code is self explanatory.

```
#include <stdio.h>

int main ()
{
  char ch;
  ch = getchar();
  printf ("Character entered is: %c\n", ch);
  return 0;
}
```

Input:
a

Output:
Character entered is: a

Therefore getchar() can be used to read the whitespace characters from the input, only if you know the exact count of such characters.

5 For loop example : Primality check

The following code takes an integer as input and displays whether it is prime or composite. You can play with it to figure out the underlying algorithm.

```
#include<stdio.h>

int main() {
    int i, n;
    scanf("%d",&n);
    for (i = 2; i < n; i++) {
        if ((n%i) == 0) {
            break;
        }
    }
    if (i == n) {
        printf("Prime");
    } else {
        printf("Composite");
    }
    return 0;
}
```

Input:
7

Output:
Prime