Switch Statement and Functions

16 February 2022 15:27

Switch statement is a cleaner way of representing various cases for the same variable as compared to an if-else statement.

If you are repeatedly using if-else if for representing different cases of a variable like :

```
if(   ){

}
else if(   ){

}
else if(   ){

}
else if(   ){

}
:
:
else {

}
```

Then you probably should switch to switch-case statements

SYNTAX:

```
switch(variable_name){

    case value_1 :
                 ~~~~~~~~~~
                 ~~~~~~~~~~~~~
                 break; ← optional
```

```
case value_2 :
    ~~~~~~~~~~~~
    ~~~~~~~~~~~~
    break;  <-- optional.
    .  .  .  .
default :
    ~~~~~~~~~~~~
    ~~~~~~~~~~~~
}
```

⚠ If you didn't understand the syntax, then don't worry and have patience. Everything will be made clear with the code implementation.

A simple usecase of switch-case

```
char ch = '1';

cout << endl;
switch( ch ) {

    case 1: cout << "First" <<endl;
            break;

    case '1': cout << "character one" <<endl;
            break;

    default: cout << " It is default case" <<endl;

}
```

↓ Gives

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ switch.
ch && "/Users/lovebabbar/"switch

character one
```

Here, character ch is being checked for cases 1 and '1'.

Clearly, since ch = '1', the second case's condition is fulfilled and we get "character one" as output.

If char ch = 'a', then we will go to the default case if you have made one. Thus, default is optional.

Break :

Using break is optional for each case. If break is not used, all the cases starting from the first case whose condition was fulfilled, will get executed.

If break is used, only that case will get executed whose condition is fulfilled.

→ Without break :

```
char ch = 'a';
int num = 1;

cout << endl;
switch( num ) {

    case 1: cout << "First" <<endl;
            cout << " First again " <<endl;
            //break;

    case '1': cout << "character one" <<endl;
            break;

    default: cout << " It is default case" <<endl;

}
```

We will stop here if case '1' or case 1 is executed.

↓ Gives

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ switch.
ch && "/Users/lovebabbar/"switch

First
 First again
character one
```

→ With break :

```
char ch = '1';
/// int num = 1;

cout << endl;
switch( ch ) {

    case 1: cout << "First" <<endl;
            cout << " First again " <<endl;
            break;

    case '1': cout << "character one" <<endl;
            break;

    default: cout << " It is default case" <<endl;

}
```

```
}
```

Gives

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ switch.
ch && "/Users/lovebabbar/"switch

character one
```

NESTED SWITCH - CASE EXAMPLE :

```
char ch = '1';
int num = 1;
switch(ch) {
    case 1:
        cout << "First Case" << endl;
        break;
    case '1':
        cout << "Second Case" << endl;
        switch(num) {
            case 0:
                cout << "num = 0" << endl;
                break;
            case 1:
                cout << "num = 1" << endl;
                break;
            case 2:
                cout << "num = 2" << endl;
                break;
            default:
                cout << "don't know what num is" << endl;
                break;
        }
        break;

    default:
        cout << "don't know what ch is" << endl;
}
```

Notice this break. for case `1'

Gives

```
Output.txt
1  Second Case
2  num = 1
3
```

Continue in Switch - Case :

Not possible / valid to use continue statement in switch - case because we are not going through any iterations, thus there is nothing to skip.

Mini - Calculator :

```
int a, b;

cout <<" Enter the value of a "<<endl;
cin >> a;
cout<<" Enter the value of b " <<endl;
cin >> b;
char op;
cout<<" Enter the Operation you want to perform" <<endl;
cin >> op;

switch( op ) {

    case '+':  cout << (a+b) <<endl;
               break;

    case '-': cout<< (a-b) <<endl;
               break;

    case '*': cout<< (a*b) <<endl;
               break;

    case '/': cout<< (a/b) <<endl;
               break;

    case '%': cout<< (a%b) <<endl;
               break;

    default: cout << "Please enter a valid Operation " << endl;

}
```

Homework Solution :

```
#include <iostream>
using namespace std;

int main(void)
{
    int money;
    cin >> money;

    switch (1) {
        case 1:
            cout << "No of 100 Rupee Notes = " << money/100 << endl;
            money = money % 100;

        case 2:
            cout << "No of 50 Rupee Notes = " << money/50 << endl;
            money = money % 50;

        case 3:
            cout << "No of 20 Rupee Notes = " << money/20 << endl;
            money = money % 20;

        case 4:
            cout << "No of 1 Rupee Notes= " << money << endl;
            money = money % 1; // unnecessary

    }
    return 0;
}
```

```
1   1330
```

Output.txt

```
1   No of 100 Rupee Notes = 13
2   No of 50 Rupee Notes = 0
3   No of 20 Rupee Notes = 1
4   No of 1 Rupee Notes= 10
5
```

Functions :

Functions are code blocks / programs that store a code 'template' / 'blueprint' to be used again and again without making the code bulky. Any change that has to be made

is done within the function to reflect it everywhere. Functions make code readable, less bulky, less buggy.

Eg: Find power - 2^5, 3^7, 4^6

① Write the code of power 3 times and each time, take input for a and b. (a^b)

② Write a 'power' function and call it 3 times for different values of a and b as arguments.

```
int power() {
    int a, int b;
    cin >> a >> b;

    int ans = 1;

    for(int i = 1; i<=b; i++) {
        ans = ans * a;
    }

    return ans;
}
```

Calling power(a,b) in main() function.

```
int ans = power();
cout << " answer is " << ans << endl;
```

Output :

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ functio
unction1 && "/Users/lovebabbar/"function1
12 2
 answer is 144
```

Check Even :

```
bool isEven(int a) {
    //odd
    if(a&1) {
        return 0;
    }
    else { //Even
        return 1;
    }

}
```

Find ^nC_r :

$$^{n}C_{r} = \frac{n!}{r!\,(n-r)!}$$

We will need another function that finds out factorial of a number for us.

```
int factorial(int n) {

    int fact = 1;

    for(int i = 1; i<=n; i++ ) {
        fact = fact * i;
    }

    return fact;
}

int nCr(int n, int r) {

    int num = factorial(n);

    int denom = factorial(r) * factorial(n-r);

    return num/denom;
}
```

When nCr(5,2) is called in main() function,

```
int num = factorial(5);  // n = 5 for this call

int denom = factorial(2) * factorial(3);  // r = 2

return num / denom;
```

for factorial(5), n = 5 is passed to the function and it returns 120. Similar is the case for n = 2, 3.

Counting Program :

Note : If you don't want to return anything, use void as the return type.

```cpp
void printCounting(int n) {

    for(int i=1; i<=n; i++) {
        cout<< i << " ";
    }
    cout<<endl;
}

int main() {

    int n;
    cin >> n;

    printCounting(n);


    return 0;
}
```

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ countin
unting && "/Users/lovebabbar/"counting
10
1 2 3 4 5 6 7 8 9 10
```

Check Prime :

Return type ←
void/int/float/char/bool

```cpp
// 1 -> Prime no.
// 0 -> Not a Prime no.
bool isPrime(int n) {

    for( int i = 2; i<n; i++) {
        //divide hogya h , not a prime no.
        if(n%i == 0) {
            return 0;
        }
    }

    return 1;
}
```

function parameters : variables local to this function that will receive some value from the fun. where it's called.

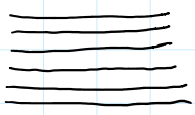
return statements

Function Call Stack :

Stack is a Last - In - First - Out `thing'. Example : Stack of plates.

Put new plates on top

Take the top plate / last plate away first.

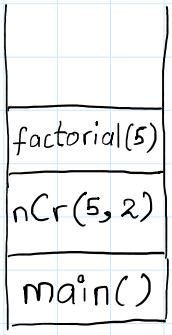


Last In

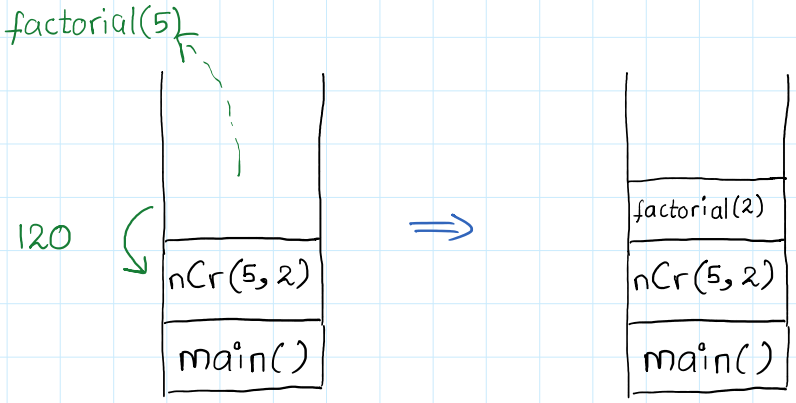
First Out

Functions that are called, get stored in a stack like structure.

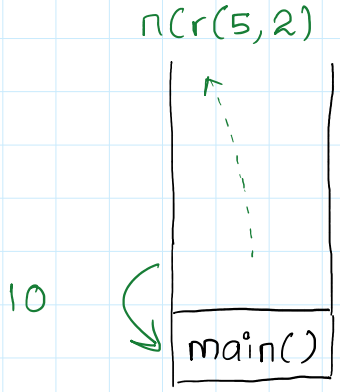
Eg: main() Call → nCr(5,2) Call → factorial(5)



factorial(5) returns 120 and then factorial(2) is called.



and so on.... till nCr(5,2) returns 10.



Homework Questions :

Q1. Print 3*n + 7 A.P.

```cpp
#include <iostream>
using namespace std;

int ap(int n) {
    return n*3 + 7;
}

int main(void)
{
    int n;
    cin >> n;
    cout << ap(n) << endl;
    return 0;
}
```

```
1  3
```

Output.txt

```
1  16
```

Q2. Print total no of SET bits in A & B.

```cpp
#include <iostream>
using namespace std;

int setBits(int num) {
    int ans = 0;
    while(num) {
        num = num & (num-1);
        ans++;
    }
    return ans;
}

int main(void)
{
    int a, b;
    cin >> a >> b;
    cout << setBits(a) + setBits(b) << endl;
    return 0;
}
```

```
1  11
2  2
```

Output.txt

```
1  4
2
```

Q3. Fibonacci Number

```cpp
#include <iostream>
using namespace std;

int fibonacciNumber(int n) {
    if(n == 1 || n == 2)
        return n-1;
    int ans = 1, a = 0, b = 1;
    for(int i=3;i<=n;i++) {
        ans = a + b;
        a = b;
        b = ans;
    }
    return ans;
}

int main(void)
{
    int n;
    cin >> n;
    cout << fibonacciNumber(n) << endl;
    return 0;
}
```

```
1  8
```

Output.txt

```
1  13
2
```

Call By Value :

When we pass a variable (not its pointer / address) to a function, we pass its value instead of passing the entire

variable (original variable). This is call by value where we pass the copy of the variable instead of the original variable present at some memory address.

If we call fibonacciNumber(6), then we are just giving the value 6 in place of n everywhere in the funct.

```
int fibonacciNumber(int n) {
    if(n == 1 || n == 2)
        return n-1;
    int ans = 1, a = 0, b = 1;
    for(int i=3;i<=n;i++) {
        ans = a + b;
        a = b;
        b = ans;
    }
    return ans;
}
```

If in main() function,

```
int n = 6;
cout << fibonacciNumber(n);
```

And change the value of n in fibonacciNumber(), then we will not change the value of the original n in the main() function.

```
#include <iostream>
using namespace std;

void fun(int n) {
    cout << "value of copy in fun() before changing = " << n << endl;
    n += 10;
    cout << "value of copy in fun() after changing = " << n << endl;
}

int main(void)
{
    int n;
    cin >> n;
    cout << "value of real n before function call = " << n << endl;
    fun(n);
    cout << "value of real n after function call = " << n << endl;

    return 0;
}
```

```
1  10
```

Output.txt

```
1  value of real n before function call = 10
2  value of copy in fun() before changing = 10
3  value of copy in fun() after changing = 20
4  value of real n after function call = 10
5
```