DEFAULT FUNCTION ARGUMENTS

```
void printline(char ='#',int =5);
int main()
{
   printline('A',10);
   printline('*');
   printline();
   return 0;
}
void printline(char ch,int n)
{
    for(int i=1;i<=n;i++)
         cout<<ch;
   cout<<endl;
```

printline('A', 10);

printline('*', 5);

printline('#', 5);

printline(43);

printline('+', 5);

X printline(, 5);

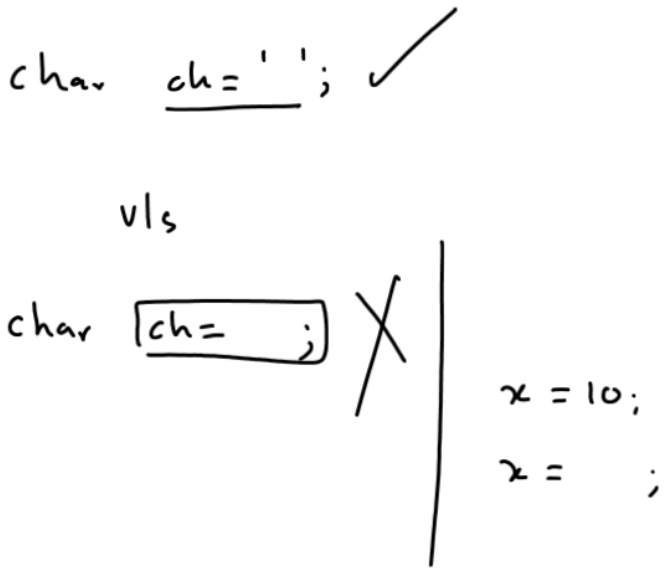
char ch;

ch = 1-9;

ch = 1;

ASCII

Cntrl: → Symbol



Default Function Arguments
==========

DFA is a technique newly introduced by C++ language which allows a programmer to call a function with fewer number of arguments as compared to it's declaration.

In other words we can say that using DFA a programmer can declare a single function and still he can call it in multiple different ways.

DFA is an easier alternate to **function overloading** because just like function overloading in DFA also we can call a function with different arguments. But in case of overloading we have to define multiple functions and only then we can call them in different ways but in case of DFA we have to define only one function and still we can call it in different ways.

Restriction On DFA

Although, DFA is a very useful technique but it has one major drawback or restriction

The drawback or restriction is that the default argument set in the function must always **trailing(ending) arguments.**

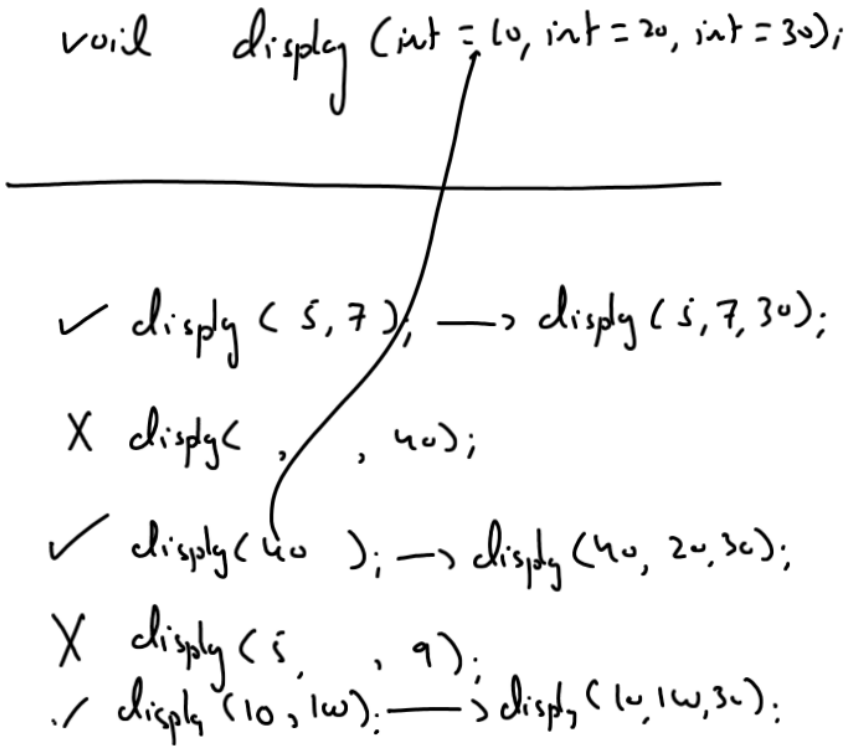
In simple words we can say that if an argument in a function has been declared as default argument then :

1. Either it should be the last argument or
2. Any argument after it must also be default.

✓ `void show ( int , int , int = 30);`

✗ `void show ( int , int = 20, int );`

✓ `void show ( int , int = 20, int = 30);`



Another point to remember about DFA is that when we call a function with default argument then if we skip any default argument then all the the arguments after it must also be skipped.