C++ for Problemm Solving

* "\n' is faster than endl.

\n : 1) It just insert a newline character into the output stream
2) It does not flush the output buffer automatically.
3) Faster, inside loops or large I/O operations.

endl : 1) Inserts a newline & flushes the output stream immediately.
2) Flushing forces the program to write everything in the buffer to the console right way.
3) slower, if done repeatedly especially (in loops).
4) Used when we ~~fran~~ want to print output immediatly, such as logging, debugging & user prompts.

Copy by Value :-

```
int f (int a, int b){
    a++;
    b++;
    return a+b;
}
int main () {
    int a, b;
    cin >>a >>b;
    cout << f (a,b);
}
```

- In this, when you pass ~~values~~ arguments to a func^n like f(a,b), you are passing copies of the values.
- changes made to a and b inside the func^n do not affect the originals in main.

call by Reference :-

```
int f (int &a , int &b){
    a++;
    b++;
    return a+b;
}
int main ( ){
    int a, b;
    cin >> a, b;
    cout<<f (a,b);
}
```

- int &a, int &b are reference variables.
- Instead of passing copies, the actual memory locations of a and b from main() are passed to f().
- So, any change inside the func^n will directly affect original a and b in main.

call By Pointer :-

```
int f2 (int *a, int *b){
    ~~a++;~~ cout<< *a <<" "<<*b <<"\n";
    ~~b++;~~
    return *a +*b;
}
int main ( ) {
    int a, b;
    cin >>a >>b;
    cout << f2 (&a, &b);
}
```

- This func^n accepts pointers to integers.
- In main, we pass the addresses of a & b using &a and &b.

Local scope : Declared inside a func^n or a block and can be used within that function.

Exa: int main(){
    int c = 5;
    int d = 10;
}
cout << 'c' <<" "<<d <<"\n";

Error : 'c' was not declared in this scope.

```
cout << c <<"  " << d << "\n";
        ^
```

'd' was not declared in this scope.

Global scope : Declared outside all func^n (including main()). It can be accessed anywhere inside the file – inside main(), other func^n etc.

How to allocate memory during runtime ?
Use dynamically memory allocation typically done using new & delete operator.

Exa :
```
int main (){
    int * num = new int (5);
    cout << *num << "\n";
}
```

output :-
```
5
```

Why ?

1. Memory size is unknown at compile time.
   → sometimes we don't know how much memory we will need until the program runs.
   → Exa : Taking user input for array size.

code
```
int n;
cin >>n;                    // User enters value
int * arr = new int[n];     // Allocate at runtime
```

2. Efficient memory usage – can allocate & deallocate memory when needed, saving space.

3. Large memory allocation

4. Data structures like linked lists, Graphs require nodes to be created during runtime.

Exa :
```
int main (){
    int *num = new int(5); // allocate memory
    cout << "num <<"\n";
    delete num; // deallocate memory
    cout <<*num << "\n";
}
```

Output
```
5
-857160145 (Garbage value)
```

Loops :-

```
string s = "abcd";
for (char ch :s) {
    cout << ch << "\n";
}
```

For each loop

→ Used when to traverse elements of container (string, vector)

→ We cannot go backward and we cannot change its value.

If we want to change ch='a' into 'b' → this cannot be psbl.

```
string s = "abcd";
for (char& ch : s) {
    cout << ch << "\n";
    ch = 'z';
}
cout << s << "\n";
```

→ So this can be used for modify char.

Output :-

```
zzzz
```

```
for (int i=0; i<5; i++) {
    cout << i << "\n";
}
```

Output :-

```
0
1
2
3
4
```

↳ This is the classic for loop