**DAY 3**

Friday, 04 July 2025

CMM level

SEICMM: Software Engineering Institute Capability Maturity Model

What is product-base and service-based company?

The **Capability Maturity Model (CMM)** describes five levels of process maturity in organizations, particularly in software development. These levels, from lowest to highest, are Initial, Repeatable, Defined, Managed, and Optimizing. Each level represents a stage of process improvement and increased control over project management and software development practices.

Here's a breakdown of each level:

* **Level 1: Initial:**

Processes are typically ad hoc, unpredictable, and chaotic. Success often depends on individual effort rather than established procedures.

* **Level 2: Repeatable:**

Basic project management processes are established, allowing for the repetition of successful practices from previous projects. Cost, schedule, and functionality can be tracked.

* **Level 3: Defined:**

Standardized and documented processes are in place for both management and engineering activities. This ensures consistency and repeatability across different projects.

* **Level 4: Managed:**

Processes are quantitatively controlled, and organizations set quantitative goals for product and process quality. Measurements and data analysis are used to manage and improve software development.

* **Level 5: Optimizing:**

Continuous process improvement is the focus, with organizations constantly seeking ways to enhance processes and introduce innovative practices.

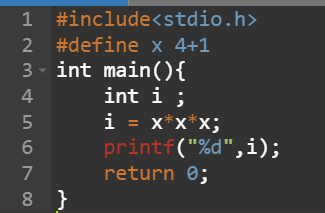
CMM provides a framework for organizations to assess their current practices and identify areas for improvement, ultimately leading to more efficient and higher-quality software development.

Zoho: <https://www.zoho.com/careers/>

**About Project**

* Project Title: Why that title
* Description
* Brief idea about project
* Team size
* To whom: Client or Users
* Platform used in the project (Frontend, Backend)
* Why that platform or lang only, deployment and other things
* Status (completed, in-progress, in future)

**C Storage classes**

#include<stdio.h>

#define x 4+1

int main(){

int i ;

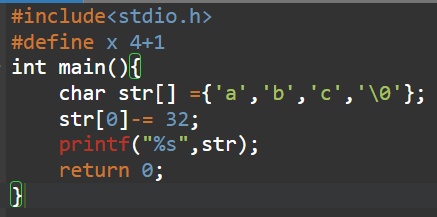
i = x\*x\*x;

printf("%d",i);

return 0;

}

* **Op = 13**

#include<stdio.h>

#define x 4+1

int main(){

char str[] ={'a','b','c','\0'};

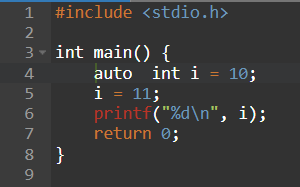
str[0]-= 32;

printf("%s",str);

return 0;

}

* **Op = Abc**

**Two storage class can’t be used of one so**

**#include <stdio.h>**

**int main() {**

**register auto int i = 10;**

**i = 11;**

**printf("%d\n", i);**

**return 0;**

**}**

🔍 Explanation:

✅ register:

Suggests to the compiler to store the variable in a CPU register for faster access.

It’s just a hint, modern compilers may ignore it.

Variables declared with register cannot have their address taken (i.e., you cannot use &i).

⚠️ auto:

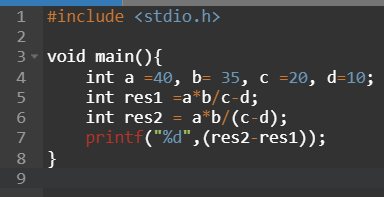
The auto keyword is completely redundant in modern C.

All local variables (inside functions) are auto by default, so auto int i = 10; is equivalent to just int i = 10;.

So, register auto int i = 10; is equivalent to just register int i = 10;.

🖨️ Output:

11

#include <stdio.h>

void main(){

int a =40, b= 35, c =20, d=10;

int res1 =a\*b/c-d;

int res2 = a\*b/(c-d);

printf("%d",(res2-res1));

}

Output: 80

**=Pointers:**

* Derived data type
* It is denoted by in-direction operator
* Types: 1] Integer pointe 2] character pointer

E.g.:

#include <stdio.h>

int main() {

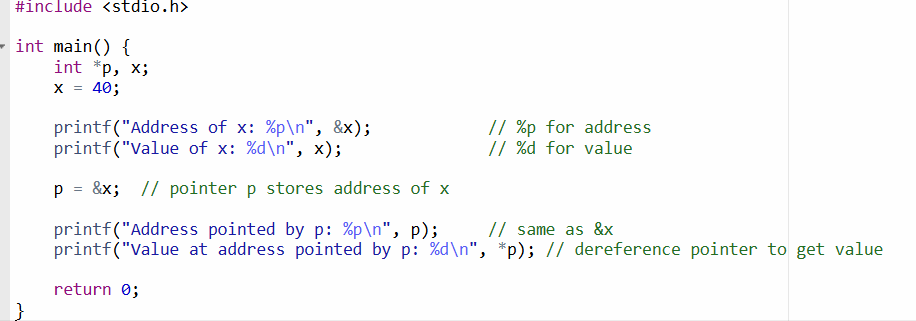
int x = 30;

printf("Value of x: %d\n", x); // prints the value: 30

printf("Address of x: %p\n", &x); // prints the memory address of x

return 0;

}

****

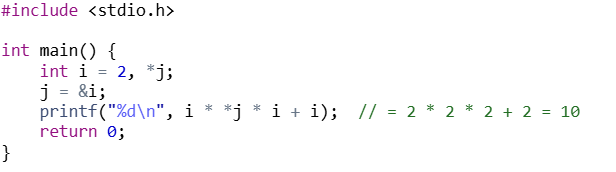
**Output:**

Address of x: 0x7ffcdf8e226c

Value of x: 40

Address pointed by p: 0x7ffcdf8e226c

Value at address pointed by p: 40



**Output:**

10

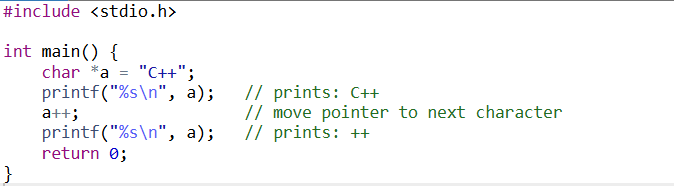
int main() {

int j = 10;

int \*i = &j;

return 0;

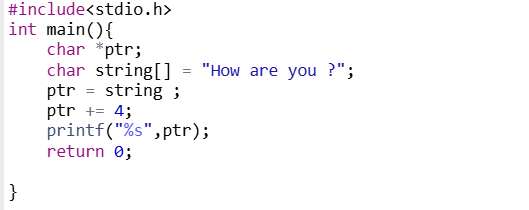
}



Output:

C++

++



**Output:**

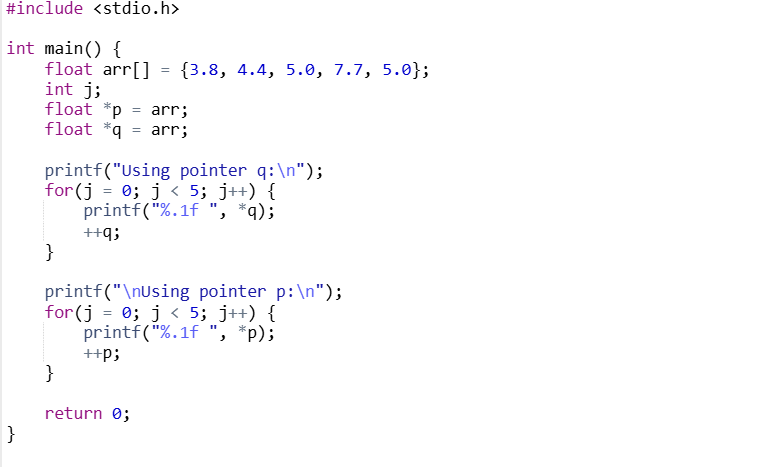
Are you?

**Slide 3: Operations Not Allowed on Pointers**

| **Operation** | **Description** |
| --- | --- |
| **Multiplication (\*p \* p)** | Only allowed when dereferencing, not for multiplying addresses |
| **Division (p / 2)** | Cannot divide a pointer |
| **Modulus (p % 2)** | Invalid: % operator not defined for pointers |
| **Incrementing beyond array bounds** | Undefined behaviour |
| **Pointer + Pointer (p1 + p2)** | Not allowed |
| **Comparing incompatible types** | Comparing int\* to float\* |

**Only valid operations:**

* p++, p-- (increment/decrement)
* p1 - p2 (if they point to the same array)
* \*p (dereferencing)
* Comparisons like p1 == p2, p1 != p2



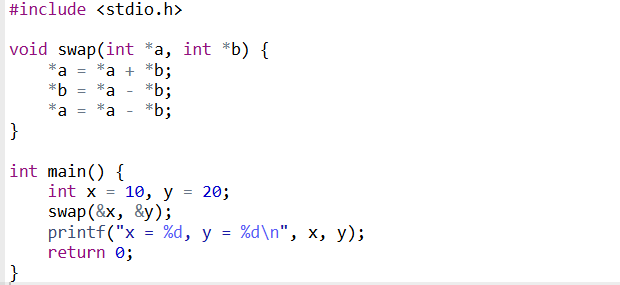
**Output:**

Using pointer q:

3.8 4.4 5.0 7.7 5.0

Using pointer p:

3.8 4.4 5.0 7.7 5.0

****

**Output:**

x = 20, y = 10