

Storage classes

CHAPTER 27



SURESH TECHS

C PROGRAMMING COURSE

```
#include<stdio.h>
int main(){
    int rank;
    printf("Welcome suresh\n");
    printf("What is your eamcet rank?");
    scanf("%d",&rank);
    if(rank<30000){
        printf("Very Good ");
    }else{
        printf("Kunchum бага chaduvu ");
    }
    printf("Your rank is : %d",rank);
    return 0;
}

void printRank(){
    printf("Rank is: %d",rank);
}
```



**Scope
Visibility
Lifetime ✓**

error: 'rank' undeclared (first use in this function)

What is the life time(జీవిత కాలం) of a human being?

- ఒకప్పుడు – 100 years
- ఇప్పుడు – 70 years
- కొన్ని సంవత్సరాలు తరువాత – 50 years
- **Life time:** The time(years) that we live in this world
- Ex: **Life time** of Suresh is **77 years**

Life time – 77 years

Narendra Modi

Scope – Palasa (Region where I am available)

Visibility – Home, Ground, Friends (Places where I am accessible within the scope)



Palasa

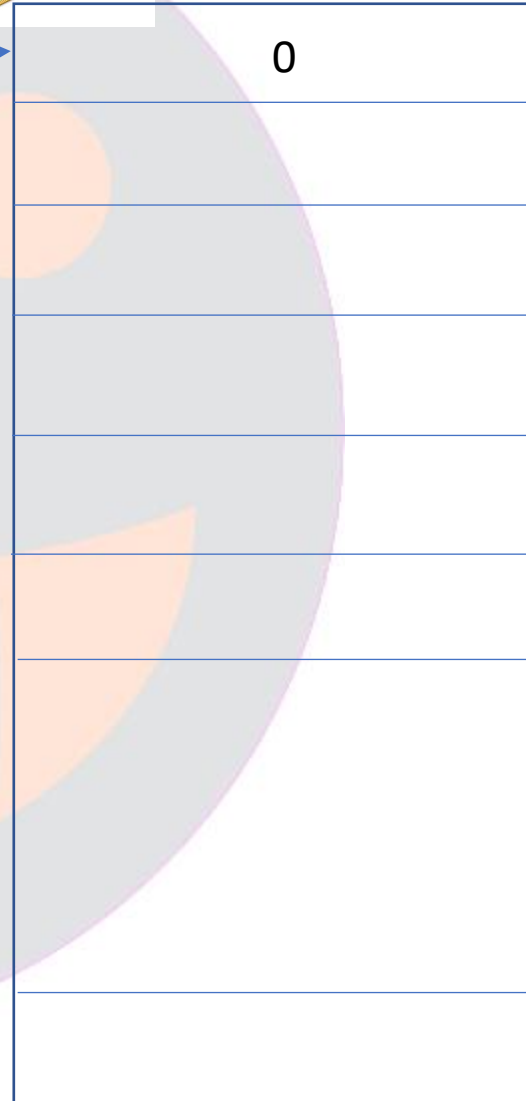
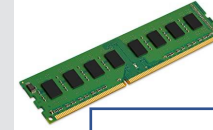
The region of the code in which the variable is available is called its **scope**

How much time the variable lives in the memory is called the **life time of the variable**

```
#include<stdio.h>
int main(){
    int rank;
    printf("Welcome suresh\n");
    printf("What is your eamcet rank?");
    scanf("%d",&rank);
    if(rank<30000){
        printf("Very Good ");
    }else{
        printf("Kunchum baga chaduvu ");
    }
    printf("Your rank is : %d",rank);
    return 0;
}

void printRank(){
    printf("Rank is: %d",rank);
}
```

If a variable is accessible or not inside a region or the program is called it's **visibility**



00110100011010110

Scope: The region in a c program where the variable is available to use

- Global scope
- Block scope
- Function scope
- Prototype scope

Visibility: Accessibility of the variable in a particular scope of the program

Lifetime: How much time the variable remains in the system memory

Storage classes defines the scope, lifetime, and visibility of the variables

Life time – 77 years

Narendra Modi

Global scope

Scope – Palasa (Region where am I available)

Visibility – Home, Ground, Friends (Places where I am accessible within the scope)



Palasa

Global scope

- Global scope variables can be **accessed from any where** in the program
- When a variable is declared **outside** of all the functions. It is then treated as **global variable**
- Global scope variables are **also known as file scope** variables
- **Memory** occupied by the variable will be **deleted** once the **program ends**

```
#include<stdio.h>
int rank;
int main(){
    printf("Welcome suresh\n");
    printf("What is your eamcet rank?");
    scanf("%d",&rank);
    if(rank<30000){
        printf("Very Good ");
    }else{
        printf("Kunchum baga chaduvu ");
    }
    printf("Your rank is : %d",rank);
    printRank();
    return 0;
}

void printRank(){
    printf("Rank is: %d",rank);
}
```


Block scope

- Variable is **only accessible within that block** not anywhere else
- A block of code can be defined in curly braces { }
- Memory occupied by the variable will be deleted once the execution of the block ends
- It is also known as **local scope**

```
#include<stdio.h>
int rank; Global scope
int main() {
    printf("Welcome suresh\n");
    printf("What is your eamcet rank?");
    scanf("%d",&rank);
    if(rank<30000) {
        printf("Very Good \n");
        → char college[]="JNTUK";
        printf("You may get: %s",college);
    } else {
        printf("Kunchum бага chaduvu ");
    }
    printf("Your rank is : %d",rank);
    printRank();
    return 0;
}

void printRank() {
    printf("Rank is: %d" rank);
}
```

```
#include<stdio.h>
int rank;
int main() {
    printf("Welcome suresh\n");
    printf("What is your eamcet rank?");
    scanf("%d",&rank);
    if(rank<30000) {
        printf("Very Good \n");
        → char college[]="JNTUK";
        printf("You may get: %s",college);
    } else {
        printf("Kunchum бага chaduvu ");
        printf("You may not get %s",college);
    }
    printf("Your rank is : %d",rank);
    printRank();
    return 0;
}

void printRank() {
    printf("Rank is: %d",rank);
}
```

What is the output of this program? When rank is 2000

Error: college undeclared

Function scope

- A variable declared inside a function has **function scope**
- **Memory** occupied by the variable will be **deleted** once the execution of the **function ends**

```
error: 'a' undeclared (first use in this function)
note: each undeclared identifier is reported only on
error: 'b' undeclared (first use in this function)
error: 'c' undeclared (first use in this function)
```

```
#include<stdio.h>
int main(){
    int a = 10;
    int b = 20;
    int c = 40;
    sum();
    return 0;
}

void sum(){
    int sum = a+b+c;
    printf("sum: %d",sum);
}
```

How to access a, b, c values inside sum function?

```
#include<stdio.h>
int main(){
    int a = 10;
    int b = 20;
    int c = 40;
    sum();
    return 0;
}

void sum(){
    int sum = a+b+c;
    printf("sum: %d",sum);
}
```

```
#include<stdio.h>
int main(){
    int a = 10;
    int b = 20;
    int c = 40;
    sum(a,b,c);
    return 0;
}

void sum(int a, int b, int c){
    int sum = a+b+c;
    printf("sum: %d",sum);
}
```

Arguments

Parameters

Function prototype scope

Function prototype scope

- Function prototype scope of variables in C is **declared in some function as its parameters**.
- These variables are **similar** to the **function scope variables** where a variable's **memory** gets deleted once the function **execution terminates**.

```
#include<stdio.h>
int main() {
    int a = 10;
    int b = 20;
    int c = 40;
    sum(a,b,c);
    sum(a,b,c);
    return 0;
}

void sum(int a, int b, int c) {
    int sum = a+b+c;
    a = 100;
    b = 100;
    printf("sum: %d\n", sum);
}
```


Scope: The region in a c program where the variable is available to use

- Global scope
- Block scope
- Function scope
- Prototype scope

Visibility: Accessibility of the variable in a particular scope of the program

Lifetime: How much time the variable remains in the system memory

Storage classes defines the scope, lifetime, and visibility of the variables

Tell me about variable a in the below program

```
#include<stdio.h>
```

```
int main() {
```

```
    int a = 10;
```

```
    int b = 20;
```

```
    int c = 40;
```

```
    sum(a,b,c);
```

```
    sum(a,b,c);
```

```
    return 0;
```

```
}
```

```
void sum(int a, int b, int c) {
```

```
    int sum = a+b+c;
```

```
    a = 100;
```

```
    b = 100;
```

```
    printf("sum: %d\n", sum);
```

```
}
```

type, memory size

Storage classes defines scope and lifetime of the variables

auto
register
static
extern

auto int a = 10
register int a = 10
static int a = 10
extern int a = 10

మనకు తెలియకుండానే కొన్ని చేస్తుంటాము

```
#include<stdio.h>
int main(){
    int a = 10;
    int b = 20;
    int c = 40;
    sum(a,b,c);
    sum(a,b,c);
    return 0;
}

void sum(int a, int b, int c){
    int sum = a+b+c;
    a = 100;
    b = 100;
    printf("sum: %d\n",sum);
}
```

మనకు తెలియకుండానే ఒక
storage class ని use చేసాము

auto

- **auto** is the **default storage class** for the variables defined **inside a function or a block**
- Those variables are **also called local variables**
- **local** variables defined with **auto** are accessible within the function boundary or the block in which they are defined
- **auto** variables are initialized with garbage value by default

- ✓ **Local scope**
- ✓ **Random initial value(garbage)**
- ✓ **Lifetime till the end of the block/function**

```
#include<stdio.h>

void sum(int a, int b, int c){
    int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
}

int main(){
    auto int a = 10;    int a = 10;
    int b = 20;
    int c = 40;
    sum(a,b,c);
    sum(a,b,c);
    return 0;
}
```

auto

- ✓ Local scope
- ✓ Random initial value(garbage)
- ✓ Lifetime till the end of the block/function

- All auto variables must be defined inside a block or function body.
- If you declare an auto variable with global scope then it would throw an error

```
#include<stdio.h>
auto int marks;
void sum(int a, int b, int c){
    int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
}

int main(){
    auto int a = 10;
    int b = 20;
    int c = 40;
    sum(a,b,c);
    sum(a,b,c);
    return 0;
}
```


register

- Variables of **register** storage are **initialized inside a register of the CPU**
- **Faster access** of the variables from the memory

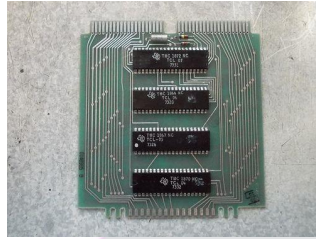


- ✓ **Local scope**
- ✓ **Random initial value(garbage)**
- ✓ **Lifetime till the end of the block/function**

```
#include<stdio.h>
void sum(int a, int b, int c){
    int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
}
```

```
int main(){
    auto int a = 10;
    register int b = 20;
    register int c = 40;
    sum(a,b,c);
    sum(a,b,c);
    return 0;
}
```

register



- Initializing variables with register storage **is just a suggestion to the compiler** by the user.
- But the **final decision is dependent on the optimization techniques** and the availability of the registers.
- **If the register is not used** the variable is stored in the main memory as in **auto** variable.

- ✓ **Local scope**
- ✓ **Random initial value(garbage)**
- ✓ **Lifetime till the end of the block/function**

```
#include<stdio.h>
void sum(int a, int b, int c){
    int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
}

int main(){
    auto int a = 10;
    register int b = 20;
    register int c = 40;
    sum(a,b,c);
    sum(a,b,c);
    return 0;
}
```

register

- Compilers are **smart enough** to detect whether a variable needs a register or not.
- Registers are **costly** and are rarely available for simple programs.



- ✓ Local scope
- ✓ Random initial value(garbage)
- ✓ Lifetime till the end of the block/function

```
#include<stdio.h>
void sum(int a, int b, int c){
    int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
}
```

```
int main(){
    auto int a = 10;
    register int b = 20;
    register int c = 40;
    sum(a,b,c);
    sum(a,b,c);
    return 0;
}
```

static

- ✓ Local scope(static local variables)
- ✓ Global scope(static global variables)

- Static variables are **initialized with zero by default**
- Static variables can be declared **globally and locally(function/block)**
- Static variables declared globally are **called static global variables**
- Static variables declared locally are **called static local variables**, and their **scope is limited to the function/block**

```
#include<stdio.h>
void sum(int a, int b, int c){
    static int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
}

int main(){
    auto int a = 10;
    register int b = 20;
    register int c = 40;
    sum(a,b,c);
    sum(a,b,c);
    return 0;
}
```

static

- ✓ Local scope(static local variables)
- ✓ Global scope(static global variables)
- ✓ Lifetime till the end the program

- Once a static variable is defined it **lives until the end of the program**
- We can create **global static variables** by defining them at the **start of the program** which can be accessed anywhere in the program

```
#include<stdio.h>
void sum(int a, int b, int c){
    static int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
}

int main(){
    auto int a = 10;
    register int b = 20;
    register int c = 40;
    sum(a,b,c);
    sum(a,b,c);
    return 0;
}
```

```
Total sum: 80
Total sum: 150
```


static

- Once a static variable is defined it **lives until the end of the program**
- We can create **global static variables** by defining them at the **start of the program** which can be accessed anywhere in the program

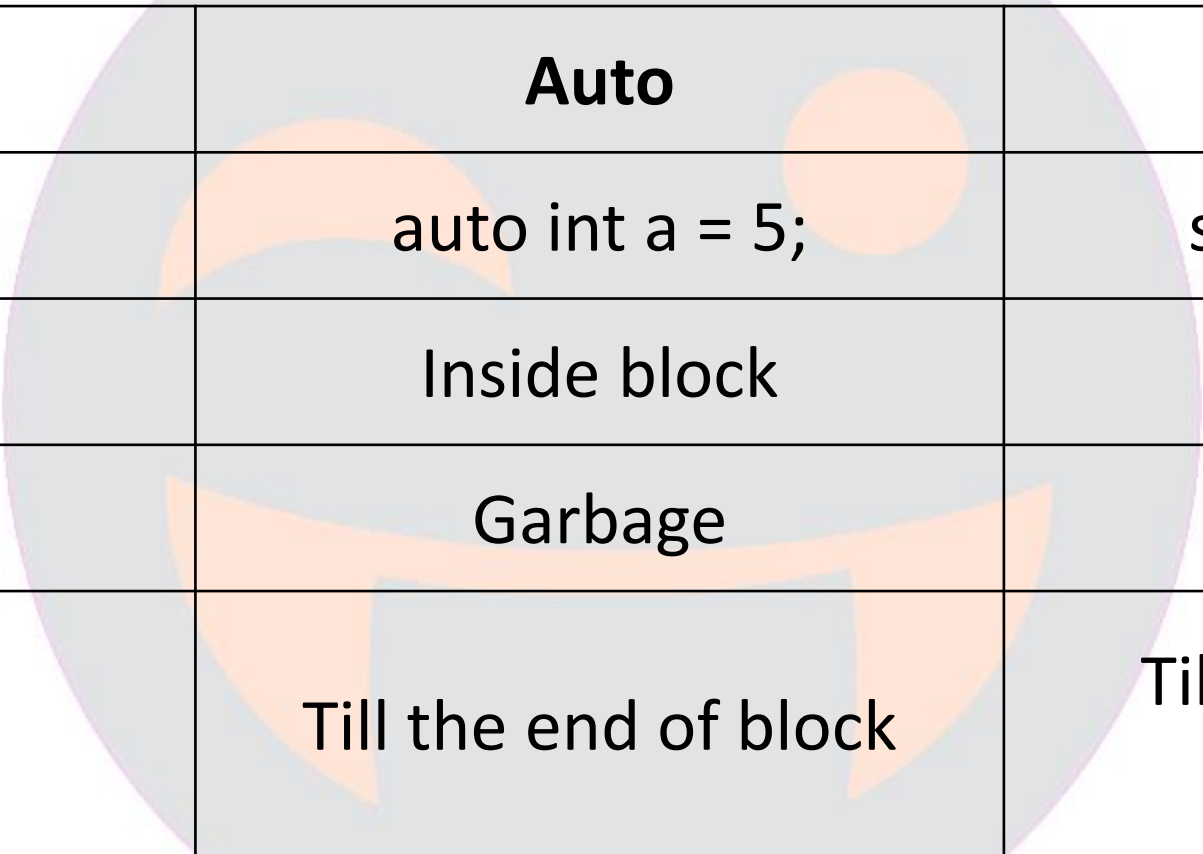
What is the output of this program?

- ✓ Local scope(static local variables)
- ✓ Global scope(static global variables)
- ✓ Lifetime till the end of the program

```
#include<stdio.h>
static int marks;
void sum(int a, int b, int c){
    static int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
    printf("Marks: %d\n",marks);
}

int main(){
    auto int a = 10;
    register int b = 20;
    register int c = 40;
    sum(a,b,c);
    marks=300;
    sum(a,b,c);
    return 0;
}
```

```
Total sum: 80
Marks: 0
Total sum: 150
Marks: 300
```

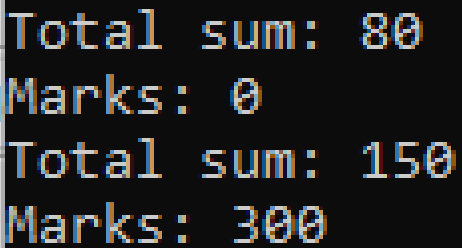


	Auto	Static
Syntax	auto int a = 5;	static int a = 5;
Scope	Inside block	Inside block
Initial Value	Garbage	Zero
Lifetime	Till the end of block	Till the end of the program

What is the output of below program?

```
#include<stdio.h>
int marks;
void sum(int a, int b, int c){
    static int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
    printf("Marks: %d\n",marks);
}

int main(){
    auto int a = 10;
    register int b = 20;
    register int c = 40;
    sum(a,b,c);
    marks=300;
    sum(a,b,c);
    return 0;
}
```



```
Total sum: 80
Marks: 0
Total sum: 150
Marks: 300
```

What is the output of below program?

```
#include<stdio.h>
int marks;
void sum(int a, int b, int c){
    static int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
    printf("Marks: %d\n",marks);
}

int main(){
    auto int a = 10;
    register int b = 20;
    register int c = 40;
    sum(a,b,c);
    int marks;
    marks=300;
    sum(a,b,c);
    return 0;
}
```

```
Total sum: 80
Marks: 0
Total sum: 150
Marks: 0
```

What is the output of below program?

```
#include<stdio.h>
int marks;
void sum(int a, int b, int c){
    static int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
    printf("Marks: %d\n",marks);
}

int main(){
    auto int a = 10;
    register int b = 20;
    register int c = 40;
    sum(a,b,c);
    extern int marks;
    marks=300;
    sum(a,b,c);
    return 0;
}
```

```
Total sum: 80
Marks: 0
Total sum: 150
Marks: 300
```

Then, what is the point of using extern, we can simply remove that line?

extern

- Extern keyword is **used declare a global variable**
- What is the **output** of the program?

Error

```
error: 'marks' undeclared (first use in this function)
```

- ✓ **Global multiple files scope**
- ✓ **Lifetime till the end of the program**

```
#include<stdio.h>
void sum(int a, int b, int c){
    static int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
    printf("Marks: %d\n",marks);
}

int main(){
    auto int a = 10;
    register int b = 20;
    register int c = 40;
    sum(a,b,c);
    extern int marks;
    marks=300;
    sum(a,b,c);
    return 0;
}

int marks=200;
```

They are mainly designed to export global variables from another file

```
#include<stdio.h>
void sum(int a, int b, int c){
    static int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
    printf("Marks: %d\n",marks);
}

int main(){
    auto int a = 10;
    register int b = 20;
    register int c = 40;
    sum(a,b,c);
    extern int marks;
    marks=300;
    sum(a,b,c);
    return 0;
}

int marks=200;
```

```
Total sum: 80
Marks: 200
Total sum: 150
Marks: 300
```

```
#include<stdio.h>
void sum(int a, int b, int c){
    static int totalSum = 10;
    int sum = a+b+c;
    totalSum = totalSum + sum;
    printf("Total sum: %d\n",totalSum);
    extern int marks;
    printf("Marks: %d\n",marks);
}

int main(){
    auto int a = 10;
    register int b = 20;
    register int c = 40;
    sum(a,b,c);
    extern int marks;
    marks=300;
    sum(a,b,c);
    return 0;
}

int marks=200;
```

Yes we can have multiple files in a project

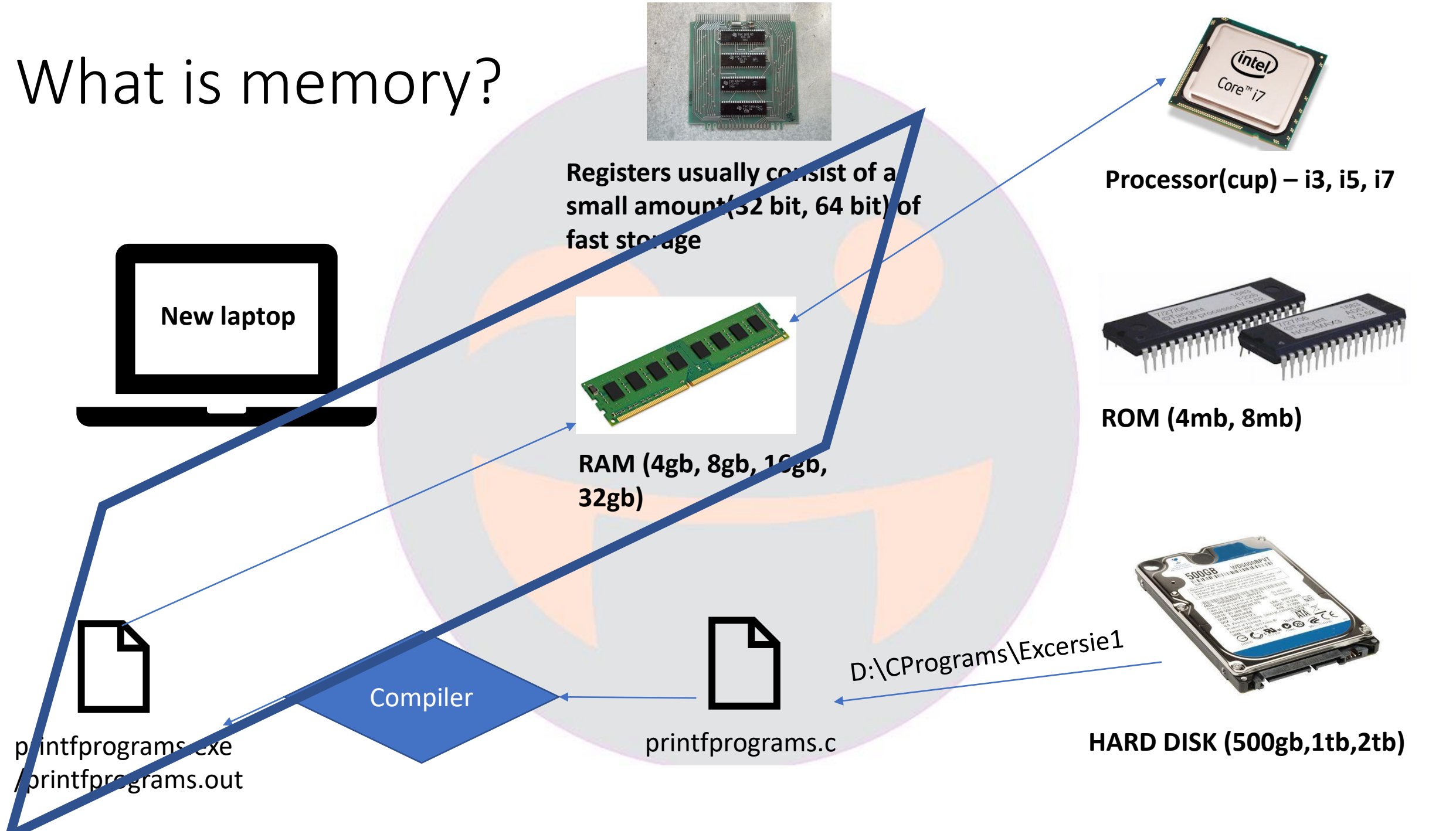


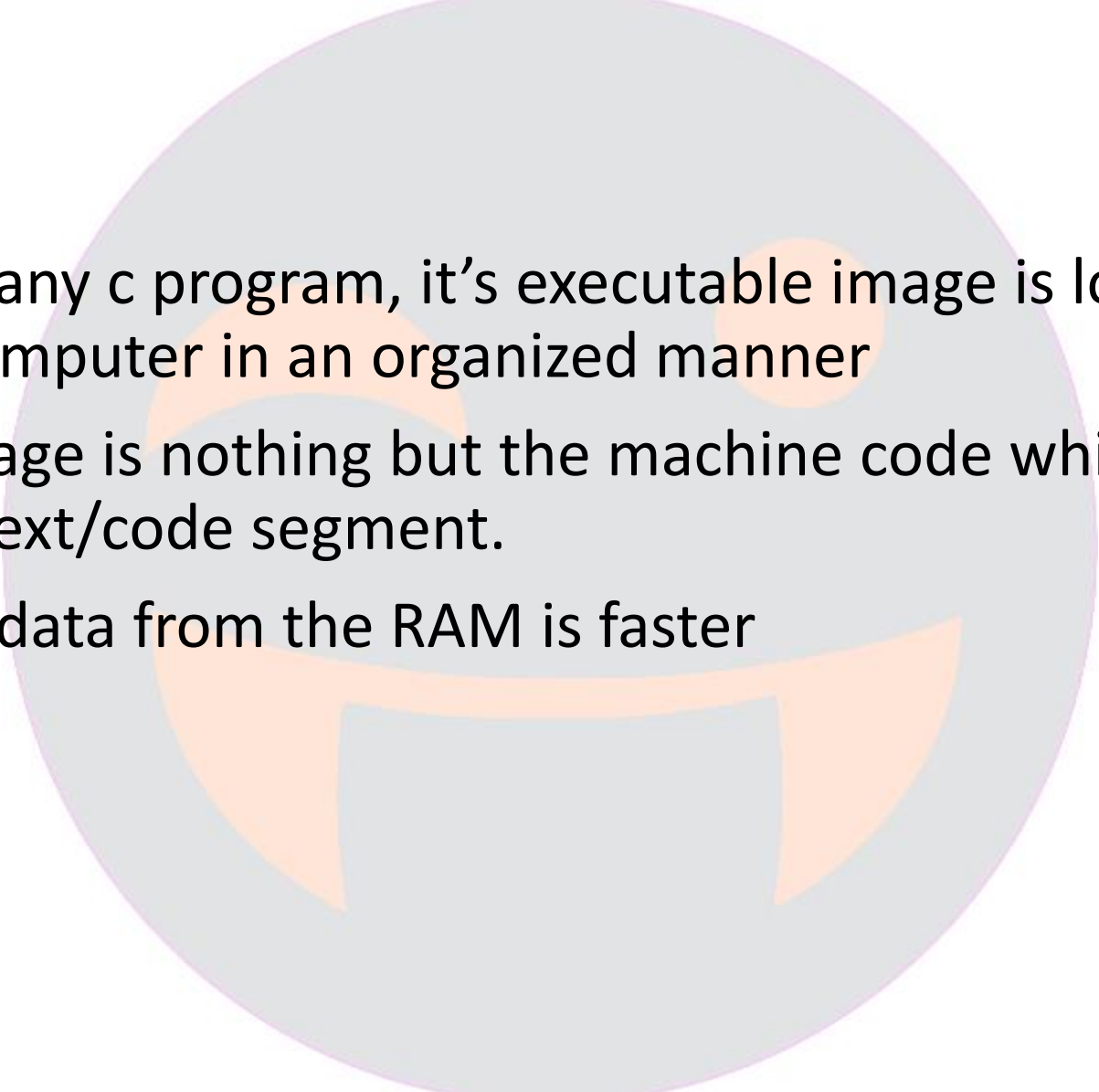
	Register	Extern
Syntax	register int a = 5;	extern int a = 5;
Scope	Inside block	Global and across files
Initial Value	Garbage	Zero
Storage location	Stored in registers	Stored in RAM
Lifetime	Till the end of block	Till the end of the program

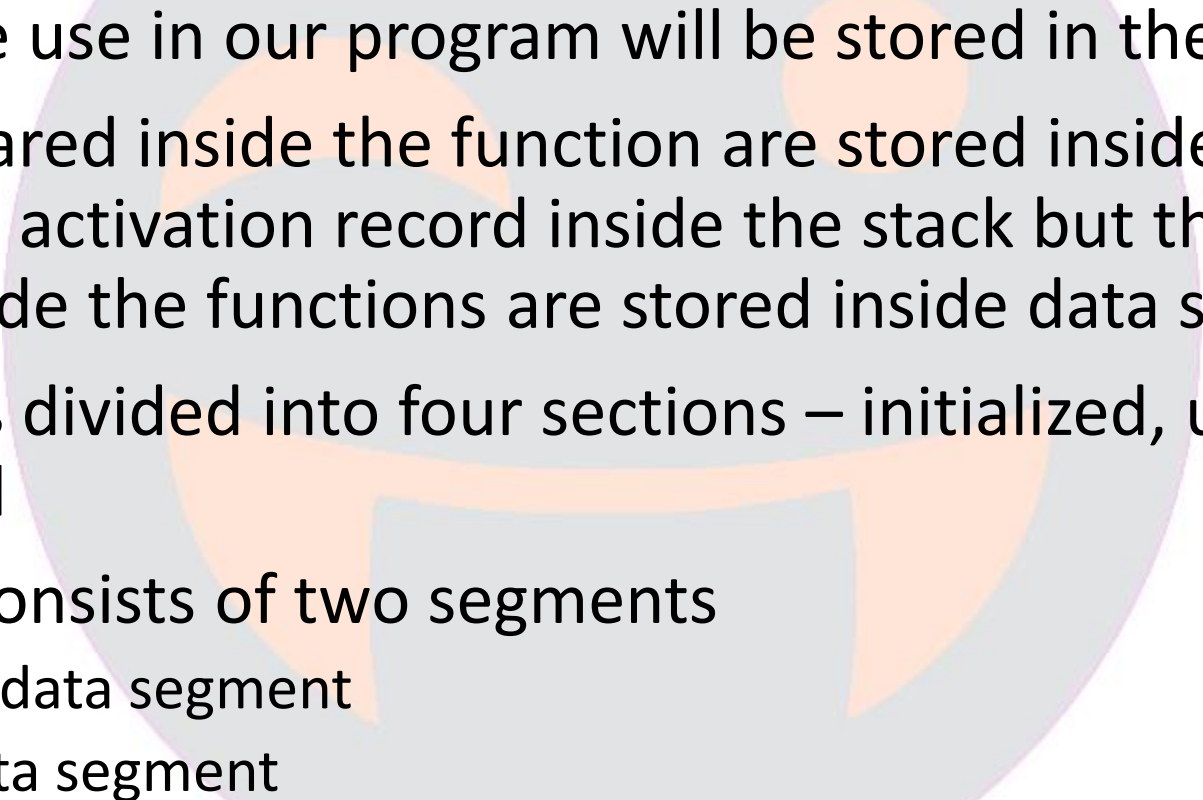
Storage classes in c

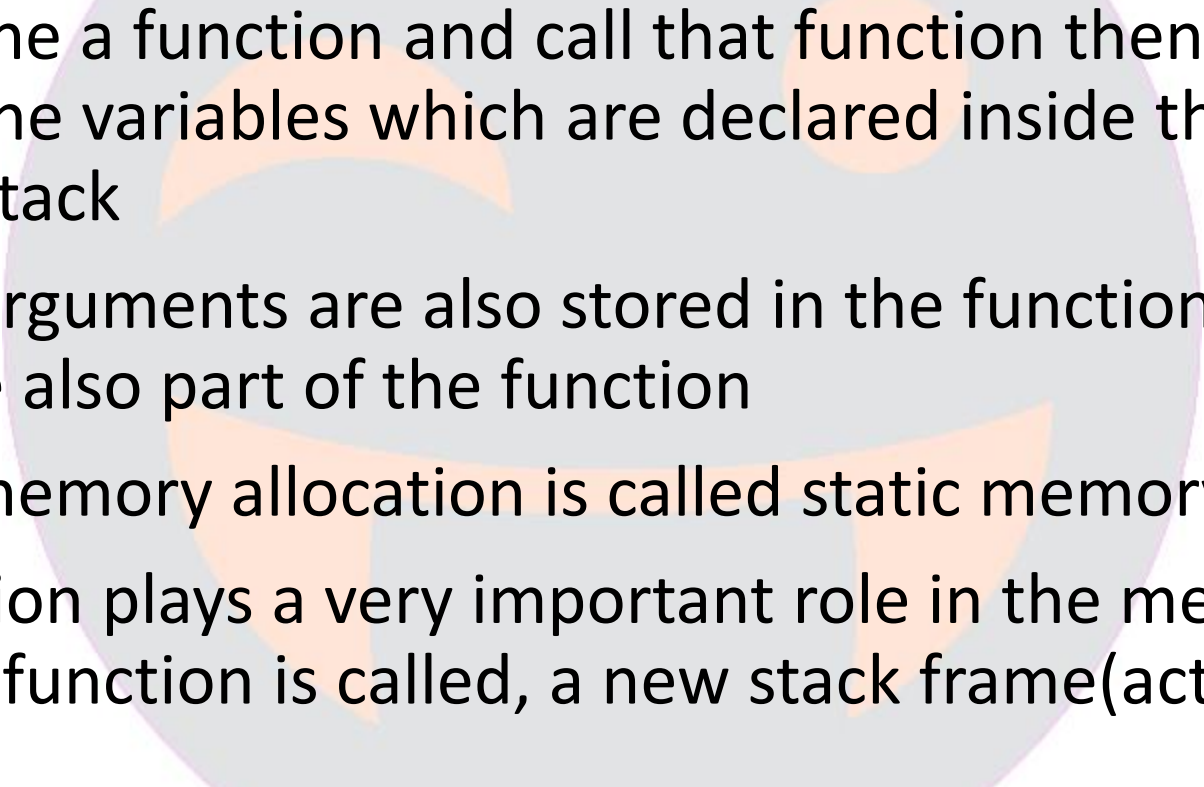
Storage specifier	Storage	Initial value	Scope	Lifetime
Auto	Stack	Garbage	Within the block	End of the block
Extern	Data segment	Zero	Global multiple files	Till end of the program
Static	Data segment	Zero	Within the block	Till end of the program
Register	CPU Register	Garbage	Within the block	End of the block

What is memory?



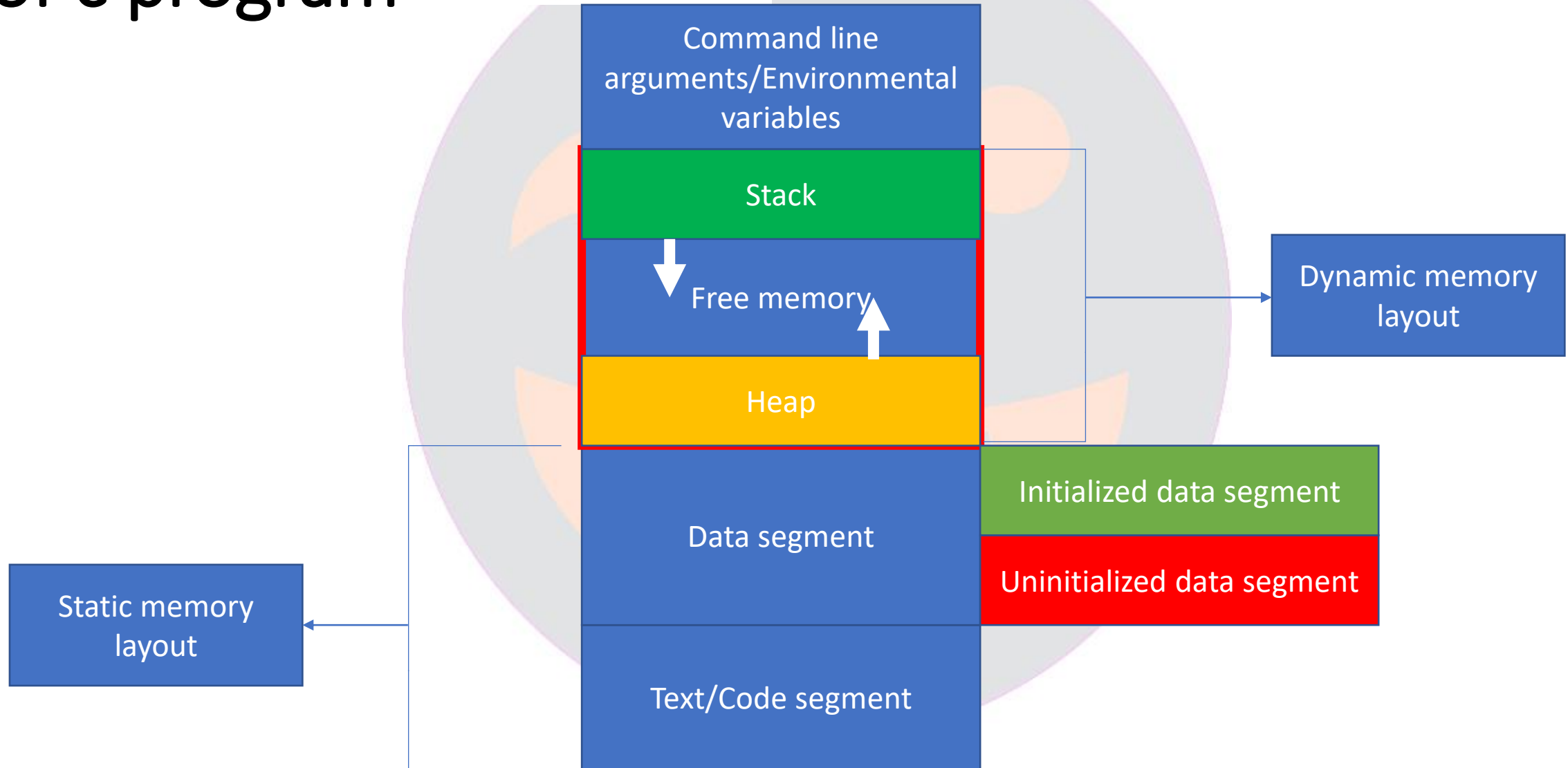
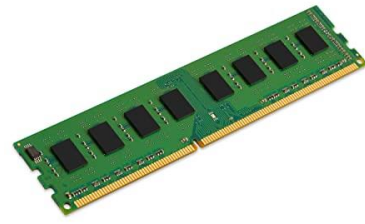
- 
- When we run any c program, it's executable image is loaded into the RAM of the computer in an organized manner
 - Executable image is nothing but the machine code which get's stored in the RAM's text/code segment.
 - Accessing the data from the RAM is faster

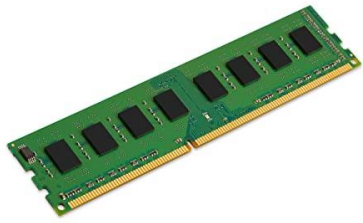
- 
- Data which we use in our program will be stored in the data section
 - Variables declared inside the function are stored inside the corresponding activation record inside the stack but the variables declared outside the functions are stored inside data segment
 - Data section is divided into four sections – initialized, uninitialized, global, or local
 - Data section consists of two segments
 - Uninitialized data segment
 - Initialized data segment

- 
- When we define a function and call that function then we use the stack frame. The variables which are declared inside the function are stored in the stack
 - The function arguments are also stored in the function as the arguments are also part of the function
 - Such type of memory allocation is called static memory allocation
 - The stack section plays a very important role in the memory because whenever the function is called, a new stack frame(activation record) is created

- Stack is also used for recursive functions. When the function is called itself again and again inside the same function which causes the stack overflow condition and it leads to the segmentation fault error in the program.

Memory layout of c program





Command line
arguments/Environmental
variables

Stack segment

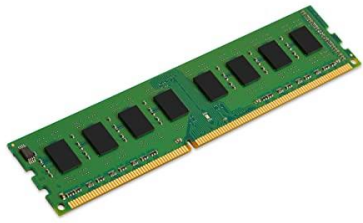
Heap segment

Data segment

Text/Code segment

```
#include<stdio.h>
int main(){
    float percentage = 94.23;
    printf("Result: %f",percentage);
    return 0;
}
```

main()
percentage = 94.23



Command line
arguments/Environmental
variables

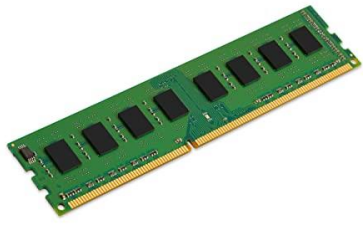
Stack segment

Heap segment

Data segment

Text/Code segment

```
#include<stdio.h>
int main(){
    float percentage = 94.23;
    printf("Result: %f",percentage);
    return 0;
}
```



Command line
arguments/Environmental
variables

Stack segment

Heap segment

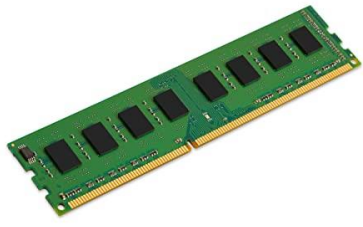
Data segment

Text/Code segment

sum()
a=10, b = 30

main()
result

```
#include<stdio.h>
int sum() {
    int a = 10;
    int b = 30;
    return a+b;
}
int diff() {
    int a = 10;
    int b = 30;
    return a-b;
}
int main() {
    int result;
    int sum1 = sum();
    int diff1 = diff();
    result = sum1*diff1;
    printf("sum: %d\n",sum1);
    printf("diff: %d\n",diff1);
    printf("result: %d",result);
    return 0;
}
```



Command line
arguments/Environmental
variables

Stack segment

Heap segment

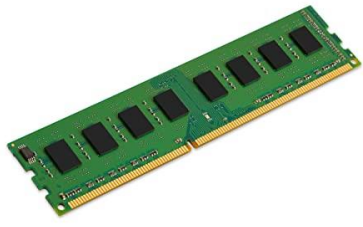
Data segment

Text/Code segment

diff()
a=10, b = 30

main()
result, sum1=40

```
#include<stdio.h>
int sum() {
    int a = 10;
    int b = 30;
    return a+b;
}
int diff() {
    int a = 10;
    int b = 30;
    return a-b;
}
int main() {
    int result;
    int sum1 = sum();
    int diff1 = diff();
    result = sum1*diff1;
    printf("sum: %d\n",sum1);
    printf("diff: %d\n",diff1);
    printf("result: %d",result);
    return 0;
}
```



Command line
arguments/Environmental
variables

Stack segment

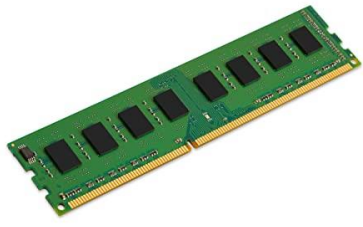
Heap segment

Data segment

Text/Code segment

main()
result, sum1=40, diff1=-20

```
#include<stdio.h>
int sum() {
    int a = 10;
    int b = 30;
    return a+b;
}
int diff() {
    int a = 10;
    int b = 30;
    return a-b;
}
int main() {
    int result;
    int sum1 = sum();
    int diff1 = diff();
    result = sum1*diff1;
    printf("sum: %d\n", sum1);
    printf("diff: %d\n", diff1);
    printf("result: %d", result);
    return 0;
}
```



Command line
arguments/Environmental
variables

Stack segment

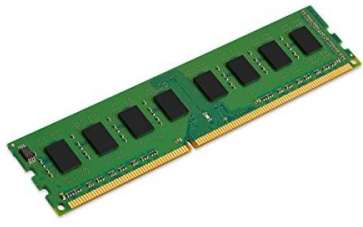
Heap segment

Data segment

Text/Code segment

main()
result=-800, sum1=40, diff1=-
20

```
#include<stdio.h>
int sum() {
    int a = 10;
    int b = 30;
    return a+b;
}
int diff() {
    int a = 10;
    int b = 30;
    return a-b;
}
int main() {
    int result;
    int sum1 = sum();
    int diff1 = diff();
    result = sum1*diff1;
    printf("sum: %d\n", sum1);
    printf("diff: %d\n", diff1);
    printf("result: %d", result);
    return 0;
}
```



Command line
arguments/Environmental
variables

Stack segment

Heap segment

Data segment

Text/Code segment

```
#include<stdio.h>
int sum() {
    int a = 10;
    int b = 30;
    return a+b;
}
int diff() {
    int a = 10;
    int b = 30;
    return a-b;
}
int main() {
    int result;
    int sum1 = sum();
    int diff1 = diff();
    result = sum1*diff1;
    printf("sum: %d\n",sum1);
    printf("diff: %d\n",diff1);
    printf("result: %d",result);
    return 0;
}
```


What next?

- Recursion

