

Memory layout of C program.

The memory layout of C program refers to how a program is organized when it is loaded into memory (or while its running)

Text (code) segment :-

- It stores compiled machine instructions (Executable code)
- It is read only

Example:-

```
#include <stdio.h>

void sum(int x, int y)
{ printf(" %d", x+y);
}

int main()
{ sum(5, 10);
  return 0;
}
```

machine instruction
011

9999 (High Memory)

Command line args
Environment variables

Stack

Grows downward

Free Memory

Heap

Grows upward

BSS segment

Data segment

a, b

Text (code) segment

code

0000 low memory

BSS Segment :-

- It stores global & static variables that are not initialized.

Example:- #include <stdio.h>

```
int a;
static int b;

int main()
{ printf(" a = %d, b = %d", a, b);
  return 0;
}
```

both are global variables
but b static

Data Segment :-

- It stores global and static variables that are initialized

Example:-

```
#include <stdio.h>

int a = 10;
static int b = 20;

int main()
{ printf(" a is %d, b is %d", a, b);
  return 0;
}
```

global variable

Heap:-

- It is used for Dynamic Memory allocation

malloc()
calloc()
realloc()

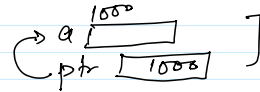
- Dynamic memory allocation is done using pointers & should be deleted manually

Example:-

```
int *ptr = malloc(sizeof(int)); // 4 Byte
OR
int *ptr = malloc(4);
```

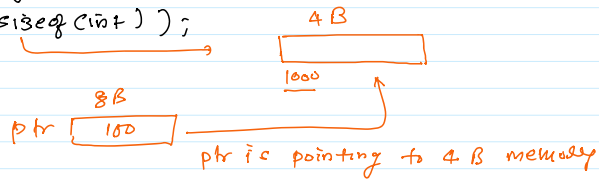
OR
`int *ptr = malloc(4);`

`int a;`
`int *ptr = &a;`



In case of Dynamic Memory allocation pointer will point to memory location without using any variable's address

`int *ptr = malloc(sizeof(int));`



Stack:-

- It stores local variable (initialized or uninitialized)
- It stores function argument
- It stores return address

Example:-
`#include <stdio.h>`
`void add(int, int);`
`int main()`

`{`
`int a;`
`int b;`
`}`

`void add(int x, int y)`
`{`
`int a;`
`int b;`
`}`

All 6 variables are stored on the stack

`= x;`

Example:-

`#include <stdio.h>`

`int a = 10;` → Data Segment

`int b;` → BSS Segment

`static int c = 100;` → Data Segment

`static int d;` → BSS Segment

`void function(int x, int y)` x, y → stack

`{`
`int e;` → stack
`int f = 200;` → stack

`int *ptr = malloc(sizeof(int));` → heap, ptr → stack

`free(ptr);`

`}`

`int main()`

`{`
`int g;` → stack

`int h = 300;` → stack

`function(5, 10);`

`return 0;`

`}`

Next topic → Dynamic Memory Allocation

5

Next topic → Dynamic Memory Allocation