CS & IT ENGINEERING

C-Programming

Fundamentals

C Programming Fundamentals

Lecture No.- 06

By- Satya sir

Recap of Previous Lecture

'C' features

- Case-Sensitive
- Portable
- Robust
- Middle level
- Structured
- Procedure-Oriented
- Platform-dependant
- Simple

Topics to be Covered

- Structure of 'C' Program
- Memory Layout of 'C' Program

Topic : Memory Layout, Structure

```
/* My First C Prg */
#include<stdio.h>
#define i 5
  int x;
Void add(int, int);
Void main( )
{
  int j=10;
 Printf("j=%d", j);
  add(5,7);
}
  Void add(int P, int q)
  {
    Printf("%d", P+q);
  }
```

- /* My First C Prg */ — Documentation Section // optional
- #include<stdio.h> — Linkage (or) Header file Section // Mandatory
- #define i 5 — Macro definitions/definition section // optional
- int x; Void add(int, int); — Global Declaration Section // optional
- Void main( ) — main( ) function // Mandatory
  - { Variables, Constants, Expressions }
- Void add(int P, int q) — Sub Program Section (Procedures (or) User defined functions) // optional

Topic : Memory Layout, Structure

Documentation Section ⇒ Also called as Title Section

- Title is represented as Comment.
- Comments are Not Executed, but Compiled.
- Comments increase Understandability & Readability of Code.
- In 'C' Language Comments are Expressed in between /* . . . */

Ex: /* My First 'C' Program */

- Comments Can be written any where in Program.

Ex:
```
/* My
   First
   C
   Program */
```

Topic : Memory Layout, Structure

Linkage (or) Header File Section

Syntax: `# include < Header file Name >`

- For Library Header file

`# include " Header file Name "`

- For User defined Header file

'C' Library → Header files → [Pre defined function definitions, Pre defined Constants]

Ex:
```
# include <stdio.h> /* Standard input output Header file */
# include <graphics.h>
# include <assert.h>
# include "Sample.h"
```

Topic : Memory Layout, Structure

main( ) function

- It indicates start Point of Execution

Syntax

```
ReturnType main( arguments )
{
    Code here
}
```

Command-line → if input is required to give from Command Prompt // optional

Ex:

```
Void main( )
{
    Printf("Hello");
}
```

```
int main(int argc, char *argv)
{
    // Code
}
```

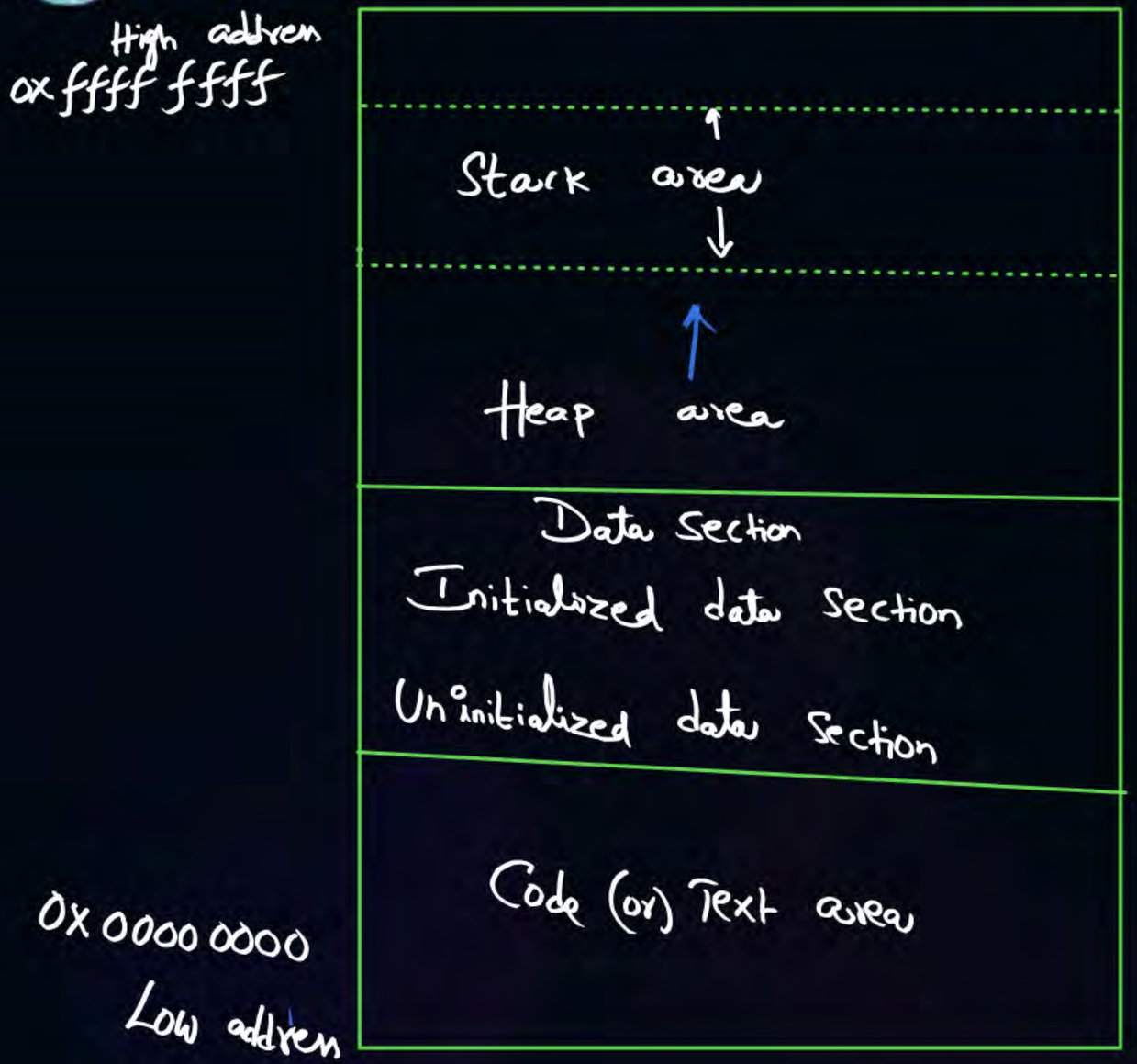
Topic : Memory Layout, Structure

Memory Layout of 'C' Program

- John Von Neumann has introduced "Stored Program Concept"
- Stored Program Concept : Not only data, but also Program Need to stored Permanently.

- The Memory area allocated for 'C' Program & data is organized into
  - High address section
  - Stack area
  - Heap area
  - Data section
  - Code section

Topic : Memory Layout, Structure



Topic : Memory Layout, Structure

Heap area

- The data that gets memory at runtime and as Per requirement

  Ex: Pointers get space in heap area

- The data that is allocated memory Using malloc(), calloc(), realloc() functions

- Stack area : Function's local variables, arguments

- Initialized data Segment : Data assigned with Values (Static Variables, global Variables, main() function Variables)

- Uninitialized data Segment : Data declared but Not assigned any Value.

  It is also called BSS

Topic : Memory Layout, Structure

- Code section or Text area ⇒ Program is stored
- High address ⇒ Environment Variables, Command line arguments.

Example

Code Section

```
#include <stdio.h>

int x;  // Uninitialized data
int y = 10; // Initialized data

void foo(int P)
{
    printf("%d", P*P);
}

void main( )
{
    int i, *k;
    char j = '@';
    foo(4);
}
```

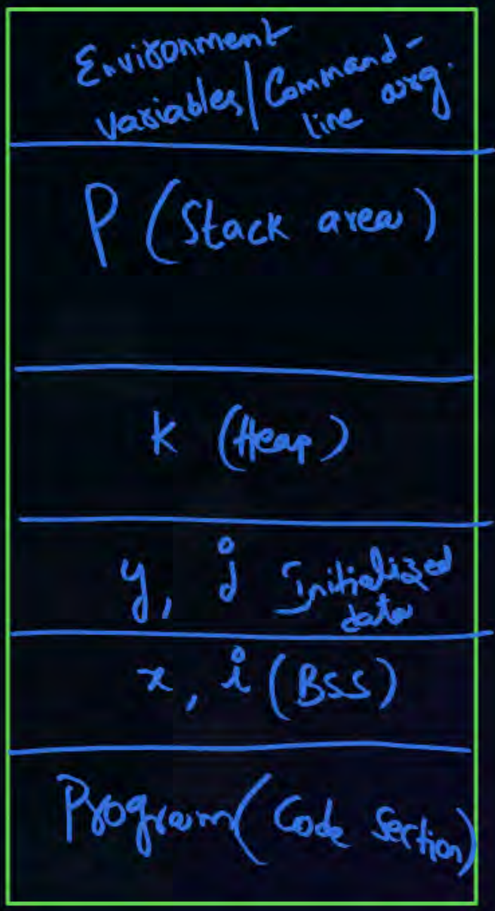
argument

Pointer

Uninitialized

Initialized

argument



2 mins Summary

- Structure of 'C' Program
- Memory Layout of 'C' Program
- Comments

THANK - YOU