Data segments in Memory

02 December 2020 09:22 AM

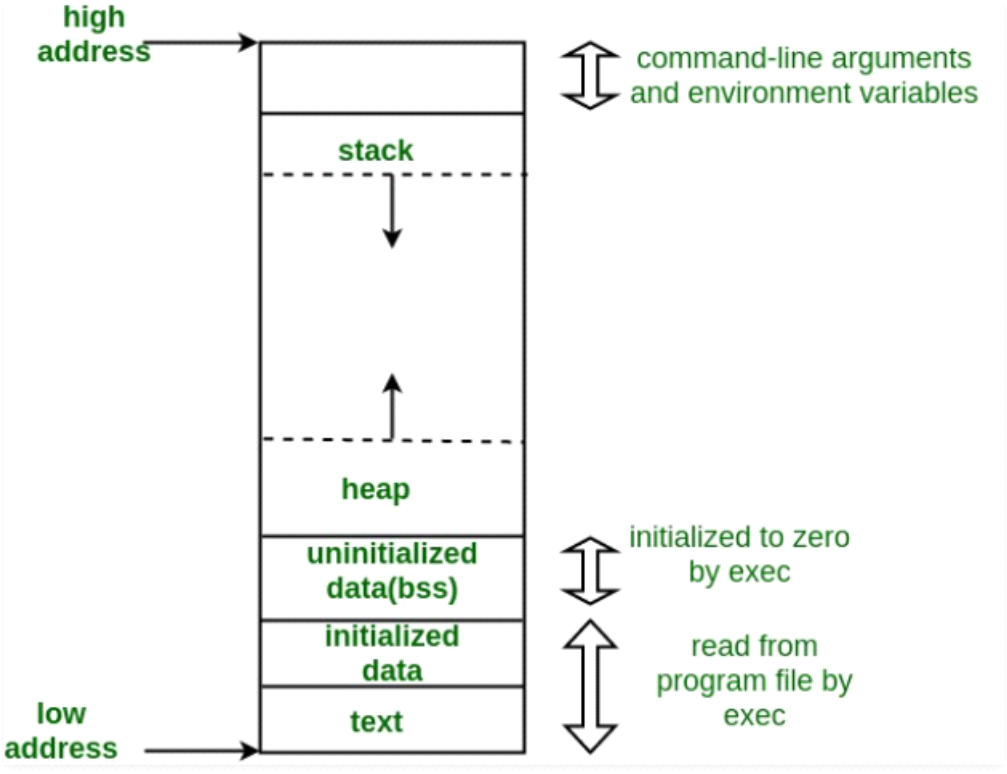
Idea

A running Program and its data is stored in the same memory space that memory is logically divided into text and data segments .

Text: this segments contains the machine code of the compiled program, this segment is read only so that it cannot be changed accidentally while program execution

Data segments: further divided into

initialize data : all the initialised variables whether global or local are stored in this segment of the memory
uninitialized data : all the uninitialized data whether global or local are stored in this segment of the memory
heap : all the dynamically allocated data are stored in this segment of the memory ,this segment heap grows upward .
stack : This segment is used for program execution and this contains all the local variables that I used inside a running block off program. hey all recursive function calls are added to stack . stack grows downward.
stacks are created hey when function call is executed All the variables local to that function are stored in the stack and when the function call terminates those variables are cleared out .



When the pointer to heap and the pointer to stack are at the same level the program throws a memory exception .

Scope of variables

17 January 2021 06:49 AM

In programming a scope is the maximal size in which no bindings change.
This means what bindings (variable names with its addresses) are used at what portion of the programme
There are to type of scoping techniques

Static scoping :

Example

```
#include<stdio.h>
int a=90;
void fun1(){
    printf("in fun1 a = %d",a);
}
void fun2(){
    int a=80;
    fun1();
}
int main(){
    fun2();
    return 0;
}
```

The output is

In fun1 a= 90

Dynamic scoping:

Example

```
#include<stdio.h>
int a=90;
void fun1(){
    printf("in fun1 a = %d",a);
}
void fun2(){
    fun1();
}
void fun3(){
    int a=30;
    fun2();
}
int main(){
```

```
    fun3();
    return 0;
}
```

the output is

In fun1 a=30 (although c follows static scoping this is just for demonstration)

Variables in c

17 January 2021 07:06 AM

Notes

- In case of variable with same name and different visibility the variable which is local is accessed
- Address operator cannot be used with a constant
- Constant qualifier
    - Variable pointer to variable int -> int * ptr
    - Const pointer to variable int -> const int *ptr
    - Const pointer to const int -> const int *const ptr

Operator precedence and associativity

17 January 2021 07:15 AM

Precedence means which operation is to be done first when there are different operator
Associativity means which operation is to be done first within same operator

Left to right : first leftmost operation then right
Right to left : first rightmost operation then left

C precedence and associativity

- Precedence of Postfix (a++) > prefix (++a)
- Postfix is L to R and prefix R to L

Shor circuiting in operators

In case of && second condition is not evaluated if first is false
In case of || second condition is not evaluated if first is true

Address arithmetic in c

17 January 2021 07:30 AM

- Two addresses can be subtracted but cannot be added , multiplied or divided
- We can add or subtract an integer no with an address
- Operators that can be used with address
    - !
    - Sizeof()
    - Type casting
    - dereferencing
    - &&,||

Flow control in c

17 January 2021 07:37 AM

For loop

```
For(exp1; exp2;exp3){
Loop body
}
```

First iteration : exp1-> exp2 -> loop body -> exp3
Second iteration : exp2 -> loop body -> exp3
Exp1,exp2,exp3 is optional
Variable cannot be declared in a for loop

- Continue transfers the control of the program to the beginning of loop

C libraries

17 January 2021 08:10 AM

C Storage class

17 January 2021 08:16 AM

Auto -

- this is storage class of local variable by default
- Initialized with a garbage value
- Gets memory at run time
- Destroyed on exit from the block

Register -

- this is a hint to compiler that this variable is frequently used
- We request compiler to store this variable in cpu register
- & (addressof) operator cannot be used
- Initialized with a garbage value
- Destroyed on exit from the block

Static -

- Compiler creates permanent storage for them
- Visible only within the block in which they are defined
- Can retain its value between function calls it is not destroyed
- If static is used in global declaration then other files cannot access it though it will be visible to the file in which it is created
- Default value of static variable is 0 or null
- **incomplete

Extern -

- this is storage class of global variable by default
- But if this key word is used explicitly no memory is allocated for that variable until it is initialized
- Global
    - int a => extern int a=0
- Extern variables can only be initialized globally only once
- A declaration of extern without initialization means the compiler will search for that variable initialized anywhere else (static or extern) if it does not find it's a compilation error
- No assignment statement globally
- If extern variable is declared globally it is visible through out the files included in that file
- Can be declared many time but must be initialized once