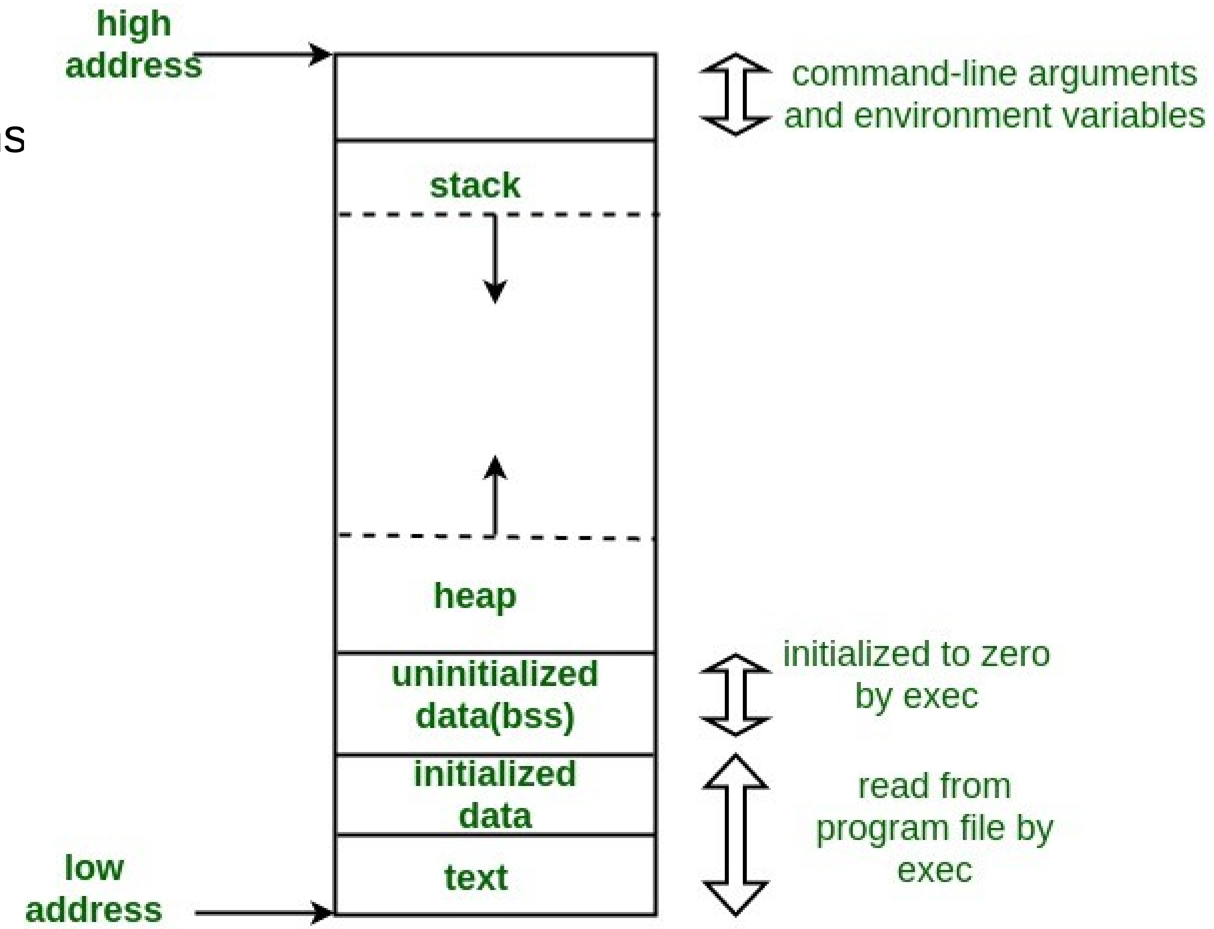
Memory Layout and Usage in C Language

Typical Memory Layout

1. Text segment (i.e. instructions
2. Initialized data segment
3. Uninitialized data segment (bss)
4. Heap
5. Stack


high address
command-line arguments and environment variables
stack
heap
uninitialized data(bss)
initialized to zero by exec
initialized data
text
read from program file by exec
low address

1. Text Segment:

A text segment, also known as a code segment or simply as text, is one of the sections of a program in an object file or in memory, which contains executable instructions.

Usually, the text segment is sharable so that only a single copy needs to be in memory for frequently executed programs, such as text editors, the C compiler, the shells, and so on.

Also, the text segment is often read-only, to prevent a program from accidentally modifying its instructions.

2. Initialized Data Segment:

A data segment is a portion of the virtual address space of a program, which contains the global variables and static variables that are initialized by the programmer.

Note that, the data segment is not read-only, since the values of the variables can be altered at run time.

This segment can be further classified into the initialized read-only area and the initialized read-write area.

The global string defined by char s[] = “hello world” in C and a C statement like int debug=1 outside the main (i.e. global) would be stored in the initialized read-write area.

static int i = 10 will be stored in the data segment and global int i = 10 will also be stored in data segment

3. Uninitialized Data Segment:

Uninitialized data segment often called the “bss” segment, named after an ancient assembler operator that stood for “block started by symbol.”

This segment is initialized by the kernel to arithmetic 0 before the program starts executing.

uninitialized data starts at the end of the data segment and contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.

```
 static int i;
 int j; (globally declared)
```

4. Stack:

The stack area contains the program stack, a LIFO structure, typically located in the higher parts of memory.

On the standard PC x86 computer architecture, it grows toward address zero; on some other architectures, it grows in the opposite direction.

A “stack pointer” register tracks the top of the stack; it is adjusted each time a value is “pushed” onto the stack.

The set of values pushed for one function call is termed a “stack frame”

Stack, where automatic variables are stored, along with information that is saved each time a function is called.

Each time a function is called, the address of where to return to and certain information about the caller’s environment, such as some of the machine registers, are saved on the stack.

5. Heap:

Heap is the segment where dynamic memory allocation usually takes place.

The heap area begins at the end of the BSS segment and grows to larger addresses from there. The Heap area is managed by malloc, realloc, and free.

The Heap area is shared by all shared libraries and dynamically loaded modules in a process.

Example: 1

```
int main(void)
{
    return 0;
}
```

```
gcc memory-layout.c -o memory-layout
```

```
size memory-layout
```

```
text      data       bss       dec       hex    filename
960        248         8      1216       4c0    memory-layout
```

Example: 2

add one global variable in the program, now check the size of bss

```
include <stdio.h>

int global; /* Uninitialized variable stored in
bss*/

int main(void)
{
    return 0;
}
```

```
text      data      bss      dec      hex   filename
 960       248       12     1220      4c4   memory-layout
```

Example: 3

add one static variable which is also stored in bss.

```
int global; /* Uninitialized variable stored in bss*/

int main(void)
{
    static int i; /* Uninitialized static variable stored in bss
*/
    return 0;
}
```

```
text       data        bss        dec        hex    filename
 960        248         16       1224        4c8    memory-layout
```

Example: 4

Initialize the static variable which will then be stored in the Data Segment (DS)

```
#include <stdio.h>

int global; /* Uninitialized variable stored in bss*/

int main(void)
{
    static int i = 100; /* Initialized static variable stored in
DS*/
    return 0;
}
```

```
text       data        bss        dec        hex    filename
960          252         12       1224         4c8    memory-layout
```

Example : 5

initialize the global variable which will then be stored in the Data Segment (DS)

```
#include <stdio.h>

int global = 10; /* initialized global variable stored in DS*/

int main(void)
{
    static int i = 100; /* Initialized static variable stored in
DS*/
    return 0;
}
```

```
text       data       bss       dec       hex    filename
960         256         8      1224        4c8    memory-layout
```