G-Fact 1 | (Sizeof is an operator)

G-Fact 2

G-Fact 3

G-Fact 4

G-Fact 5

G-Fact 6

G-Fact 7

G-Fact 8

How are variables scoped in C – Static or Dynamic?

Memory Layout of C Programs

A typical memory representation of C program consists of following sections.

1. Text segment
2. Initialized data segment
3. Uninitialized data segment
4. Stack
5. Heap

Scope rules in C

How Linkers Resolve Global Symbols Defined at Multiple Places?

Complicated declarations in C

Redeclaration of global variable in C

Data Types in C

Use of bool in C

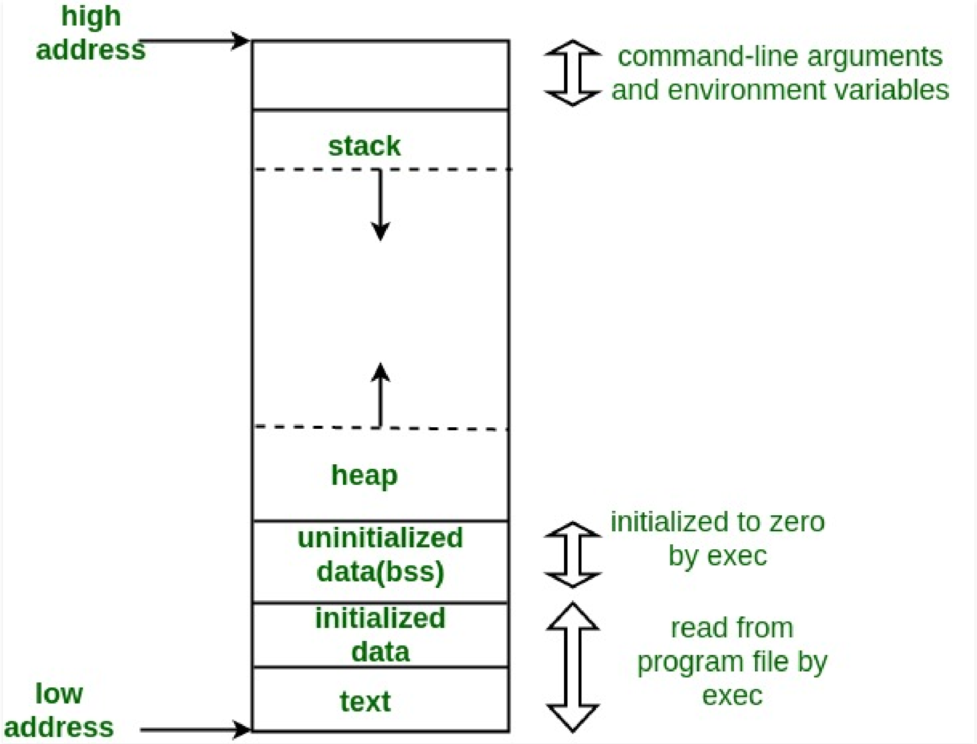
Integer Promotions in C

Comparison of a float with a value in C

Storage Classes in C

Static Variables in C

How to deallocate memory without using free() in C?


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

A typical memory layout of a running process

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.

As a memory region, a text segment may be placed below the heap or stack in order to prevent heaps and stack overflows from overwriting it.

Most popular in C

calloc() versus malloc()

How does free() know the size of memory to be deallocated?

Use of realloc()

int (1 sign bit + 31 data bits) keyword in C

Program error signals

Why array index starts from zero ?

TCP Server-Client implementation in C

How to return multiple values from a function in C or C++?

Dynamic Memory Allocation in C using malloc(), calloc(), free() and realloc()

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:
Initialized data segment, usually called simply the Data Segment. A data segment is a portion of virtual address space of a program, which contains the global variables and static variables that are initialized by the programmer.

Note that, 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 initialized read-only area and initialized read-write area.

For instance 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 initialized read-write area. And a global C statement like const char* string = “hello world” makes the string literal “hello world” to be stored in initialized read-only area and the character pointer variable string in initialized read-write area.

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

Dividing a Large file into Separate Modules in C/C++, Java and Python

Applications of Pointers in C/C++

“static const” vs “#define” vs “enum”

Pre-increment and Post-increment in C/C++

Code Optimization Technique (logical AND and logical OR)

More related articles in C

Loader in C/C++

C program to store Student records as Structures and Sort them by Name

Sum of array Elements without using loops and recursion

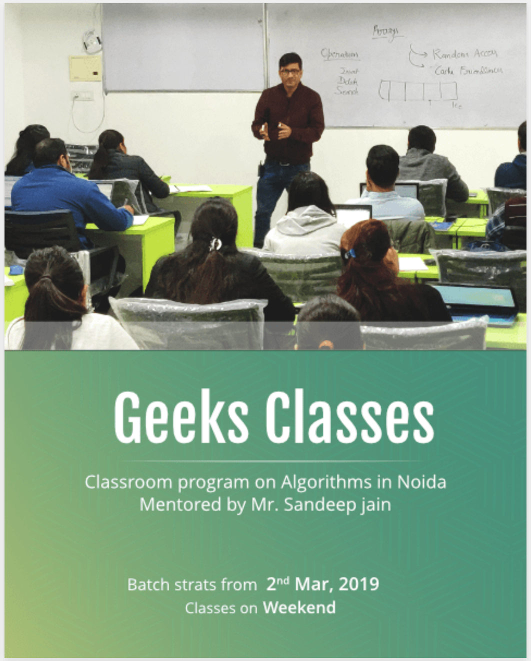
3. Uninitialized Data Segment:
Uninitialized data segment, often called the “bss” segment, named after an ancient assembler operator that stood for “block started by symbol.” Data in this segment is initialized by the kernel to arithmetic 0 before the program starts executing

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.

For instance a variable declared static int i; would be contained in the BSS segment.
For instance a global variable declared int j; would be contained in the BSS segment.

4. Stack:
The stack area traditionally adjoined the heap area and grew the opposite direction; when the stack pointer met the heap pointer, free memory was exhausted. (With modern large address spaces and virtual memory techniques they may be placed almost anywhere, but they still typically grow opposite directions.)

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 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


Geeks Classes
Classroom program on Algorithms in Noida
Mentored by Mr. Sandeep jain
Batch strats from 2nd Mar, 2019
Classes on Weekend

termed a “stack frame”; A stack frame consists at minimum of a return address.

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. The newly called function then allocates room on the stack for its automatic and temporary variables. This is how recursive functions in C can work. Each time a recursive function calls itself, a new stack frame is used, so one set of variables doesn’t interfere with the variables from another instance of the function.

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, which may use the brk and sbrk system calls to adjust its size (note that the use of brk/sbrk and a single “heap area” is not required to fulfill the contract of malloc/realloc/free; they may also be implemented using mmap to reserve potentially non-contiguous regions of virtual memory into the process’ virtual address space). The Heap area is shared by all shared libraries and dynamically loaded modules in a process.

Examples.

How to find Segmentation Error in C & C++ ? (Using GDB)

Difference between Call by Value and Call by Reference

GDB (Step by Step Introduction)

Difference between const int*, const int * const, and int const *

Why strcpy and strncpy are not safe to use?

Difference between C and C++

Program to copy the contents of one array into another in the reverse order

Communication between two process using signals in C

strrev() function in C

strlen() function in c

The size(1) command reports the sizes (in bytes) of the text, data, and bss segments. (for more details please refer man page of size(1))

1. Check the following simple C program

```
#include <stdio.h>

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

```
[narendra@CentOS]$ gcc memory-layout.c -o memory-layou
[narendra@CentOS]$ size memory-layout
text          data          bss          dec          hex     fi
960           248           8            1216         4c0    mem
```

2. Let us add one global variable in program, now check the size of bss (highlighted in red color).

```
#include <stdio.h>

int global; /* Uninitialized variable stored in bs

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

Program to check if two strings are same or not

Measure execution time with high precision in C/C++

Why does sizeof(x++) not increment x in C?

```
[narendra@CentOS]$ gcc memory-layout.c -o memory-layou
[narendra@CentOS]$ size memory-layout
text          data          bss          dec          hex     fi
 960           248           12         1220          4c4     me
```

3. Let us add one static variable which is also stored in bss.

```
#include <stdio.h>

int global; /* Uninitialized variable stored in bs

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

```
[narendra@CentOS]$ gcc memory-layout.c -o memory-layou
[narendra@CentOS]$ size memory-layout
text          data          bss          dec          hex     fi
 960           248           16         1224          4c8     me
```

4. Let us initialize the static variable which will then be stored in Data Segment (DS)

```
#include <stdio.h>

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

```
[narendra@CentOS]$ gcc memory-layout.c -o memory-layou
[narendra@CentOS]$ size memory-layout
text       data        bss        dec        hex    fi
960         252         12       1224        4c8    me
```

5. Let us initialize the global variable which will then be stored in Data Segment (DS)

```
#include <stdio.h>

int global = 10; /* initialized global variable st

int main(void)
{
    static int i = 100; /* Initialized static vari
    return 0;
}
```

```
[narendra@CentOS]$ gcc memory-layout.c -o memory-layou
[narendra@CentOS]$ size memory-layout
text       data        bss        dec        hex    fi
960         256          8       1224        4c8    me
```

This article is compiled by **Narendra Kangralkar**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source:
http://en.wikipedia.org/wiki/Data_segment
http://en.wikipedia.org/wiki/Code_segment
http://en.wikipedia.org/wiki/.bss
http://www.amazon.com/Advanced-Programming-UNIX-Environment-2nd/dp/0201433079

Recommended Posts:

Common Memory/Pointer Related bug in C Programs
IPC through shared memory
Memory leak in C++ and How to avoid it?
How to deallocate memory without using free() in C?
What is Memory Leak? How can we avoid?
Stack vs Heap Memory Allocation
MCQ on Memory allocation and compilation process
Memory Segmentation in 8086 Microprocessor
C | Dynamic Memory Allocation | Question 1
C | Dynamic Memory Allocation | Question 2
C | Dynamic Memory Allocation | Question 3
C | Dynamic Memory Allocation | Question 8

C | Dynamic Memory Allocation | Question 5

C | Dynamic Memory Allocation | Question 6

C | Dynamic Memory Allocation | Question 7

Article Tags : C C-Dynamic Memory Allocation system-programming

Practice Tags : C

19

□ To-do □ Done

2.6

Based on **156** vote(s)

Feedback Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Previous

|< Hiding of all overloaded methods with same name in

Next

Templates and Default >| Arguments

base class

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!

GeeksforGeeks
A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY
About Us
Careers
Privacy Policy
Contact Us

LEARN
Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

PRACTICE
Company-wise
Topic-wise
Contests
Subjective Questions

CONTRIBUTE
Write an Article
Write Interview Experience
Internships
Videos