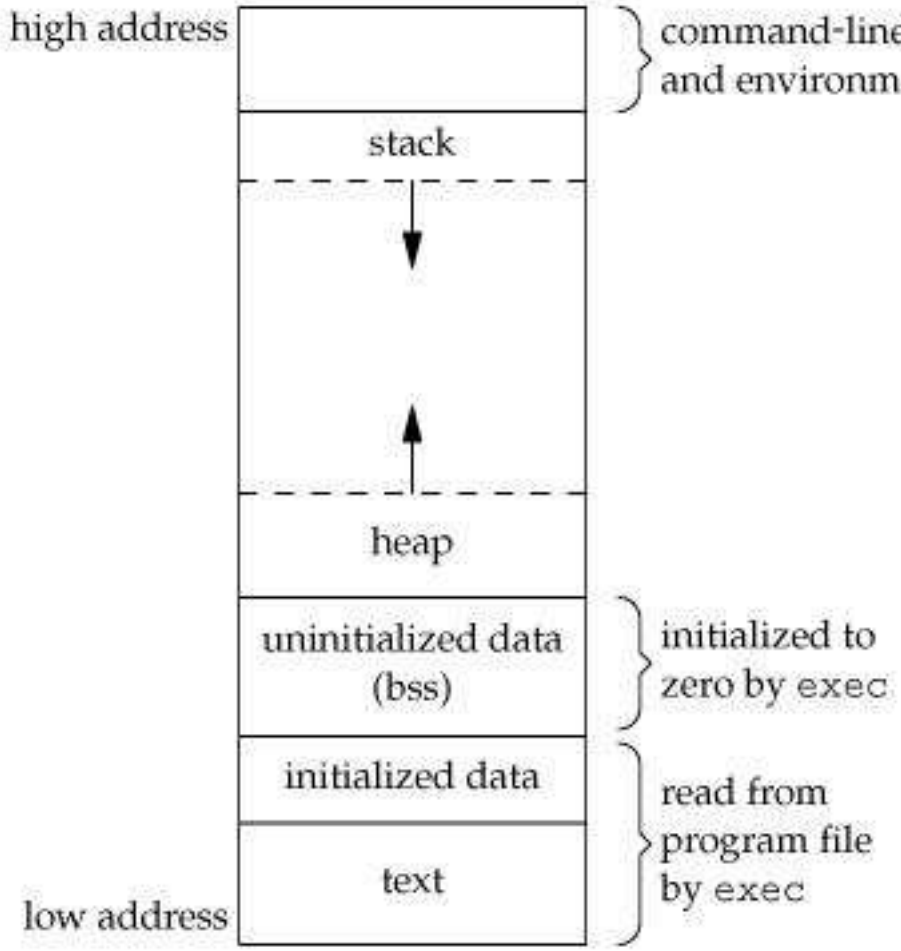
Memory Management

Week 05 – Lab 5

https://goo.gl/ofxCMH

Memory Layout of C Program



Text or Code Segment

- Contains machine code of the compiled program. The text segment of an executable object file is often read-only segment that prevents a program from being accidentally modified.

Initialised Data or Data Segment

- Stores all global, static, constant, and external variables (declared with extern keyword) that are initialised beforehand

Uninitialised Data or .bss Segment

- Stores all uninitialised global, static, and external variables (declared with extern keyword)

Stack Segment

- Stores all local variables and is used for passing arguments to the functions along with the return address of the instruction which is to be executed after the function call is over

Heap Segment

- Part of RAM where dynamically allocated variables are stored. In C language dynamic memory allocation is done by using *malloc* and *calloc* functions.

Exercise 1

- Use $*size* command to determine the size of text, data and bss segments of week 4 exercise 4 program (use any other executable file in case you don't have week4/ex4 file).

Pointers revision

- Each variable represents an address in memory and a value.
- Address: &variable = address of variable
- A pointer is a variable that “points” to the block of memory that a variable represent

Pointers revision (2)

- Declaration: data_type *pointer_name;
- Example:

  char x = 'a';

  char *ptr = &x; // ptr points to a char x
- Pointers are integer variables themselves, so can have pointer to pointers:

  char **ptr;

Pointers revision (3)

- Dereferencing = Using Addresses

```
int x = 5;

int *ptr = &x;

*ptr = 6; // Access x via ptr, and changes it to 6
 printf("%d", x); // Will print 6 now
```

Why use pointers?

- Pass-by-reference rather than value.
  void sample_func(char* str_input);

- Manipulate memory effectively.

- Useful for arrays (Array in C - a pointer and a length)

sizeof(type)

- Returns number of bytes of a data type

Exercise 2

- Write a C program that declares 5 variables: char, int, double, a pointer to int. Print sizes of declared variables using sizeof() operator

Malloc()

void *malloc(size_t size)

```
int array[10];
int *array = malloc(10*sizeof(int));
```

- malloc does not initialise the array; this means that the array may contain random or unexpected values

Calloc()

```
void *calloc(size_t nmemb, size_t size);
```

- The calloc function allocates space for an array of items and initialises the memory to zeros.

Realloc()

```
void * realloc ( void * ptr, size_t size );
```

- The realloc function changes the size of the object pointed to by ptr to the size specified by size

Free()

void free(void *ptr)

- Releases memory allocated by malloc(), calloc() or realloc()

```
int *myStuff = malloc( 20 * sizeof(int));
if (myStuff != NULL)
{
   /* more statements here */
   /* time to release myStuff */
   free( myStuff );
}
```

Exercise 3

- Write a C program that dynamically allocates memory for an array of N integers, fills the array with incremental values starting from 0, prints the array and deallocates the memory. Program should prompt the user to enter N before allocating the memory.

Exercise 4

- Complete the following code template according to the comments. The purpose of the program is to create an initial array of a user-specified size, then dynamically resize the array to a new user-specified size. Save your solution as week5/ex4.c. Link: http://goo.gl/EMpzYz

Extra exercise 1

- Write realloc function using malloc and free
- realloc() changes the size of the memory block pointed to by ptr to size bytes. The contents will be unchanged to the minimum of the old and new sizes; newly allocated memory will be uninitialized. If ptr is NULL, the call is equivalent to malloc(size); if size is equal to zero, the call is equivalent to free(ptr). Unless ptr is NULL, it must have been returned by an earlier call to malloc(), calloc() or realloc()

Extra exercise 2

- Find and fix all the code that generates Segmentation Fault

```
#include <stdio.h>
int main() {
 char **s;
 char foo[] = "Hello World";

 *s = foo;
 printf("s is %s\n",s);
 s[0] = foo;
 printf("s[0] is %s\n",s[0]);
 return(0);
}
```