Programming and Algorithms

COMP1038.PGA

Week 6 – Lecture 3: Dynamic Memory Allocation

Dr. Pushpendu Kar

Outline

- Memory layout of C program
- Dynamic Memory Allocation
  - malloc()
  - free()
  - realloc()
- Conclusion

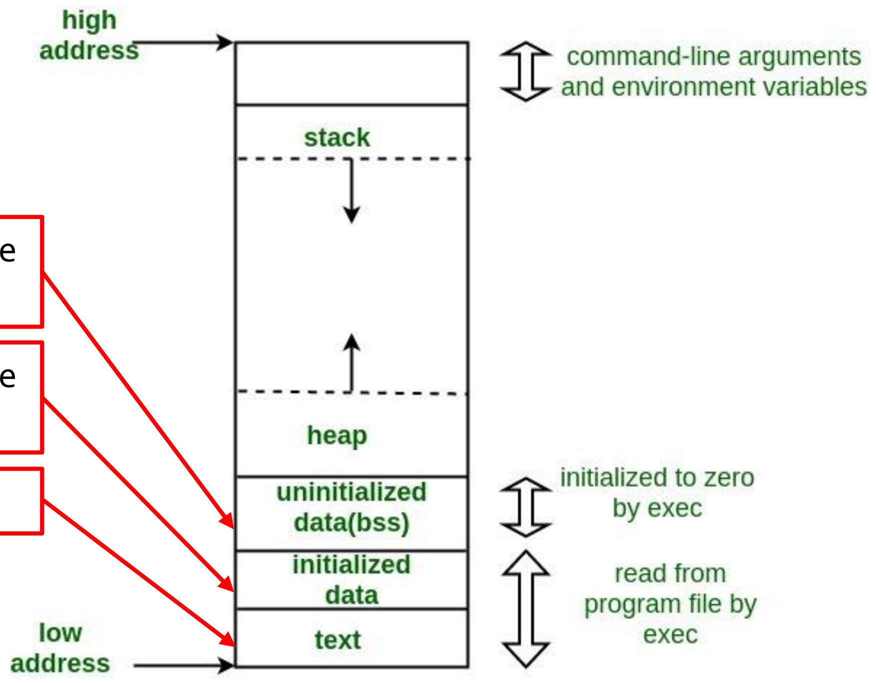
Memory Layout of C Programs

- Typical layout of a running process

Global and Static data that are not initialised.

Global and Static data that are initialised.

Code segment, executable

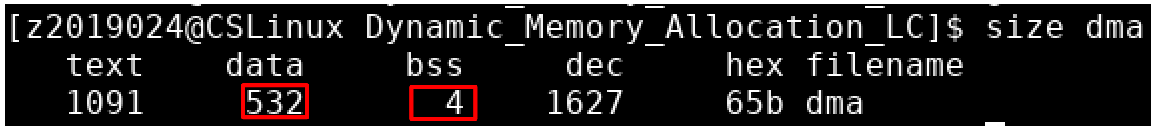

high address
command-line arguments and environment variables
stack
heap
uninitialized data(bss)
initialized to zero by exec
initialized data
read from program file by exec
text
low address

Source: https://www.geeksforgeeks.org/

Memory Layout of C Programs cont…

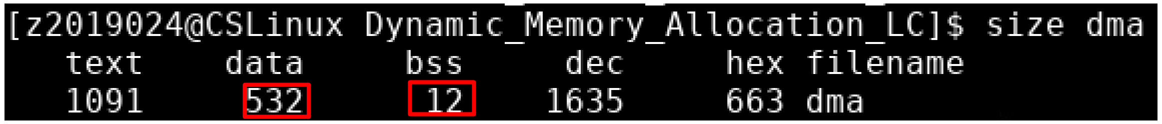
- Note the size of the **uninitialised** data (bss)

```c
#include<stdio.h>

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



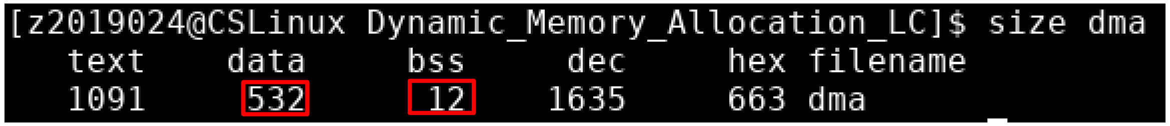
```
[z2019024@CSLinux Dynamic_Memory_Allocation_LC]$ size dma
   text    data     bss     dec     hex filename
   1091     532       4    1627     65b dma
```

```c
#include<stdio.h>

int global;

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



```
[z2019024@CSLinux Dynamic_Memory_Allocation_LC]$ size dma
   text    data     bss     dec     hex filename
   1091     532      12    1635     663 dma
```

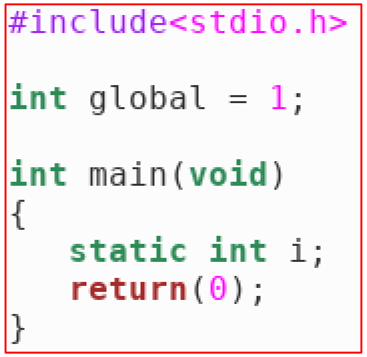
```c
#include<stdio.h>

int global;

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



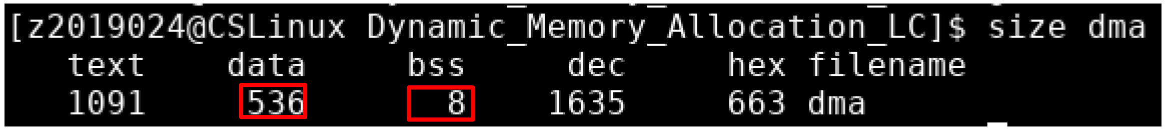
```
[z2019024@CSLinux Dynamic_Memory_Allocation_LC]$ size dma
   text    data     bss     dec     hex filename
   1091     532      12    1635     663 dma
```

Memory Layout of C Programs cont…

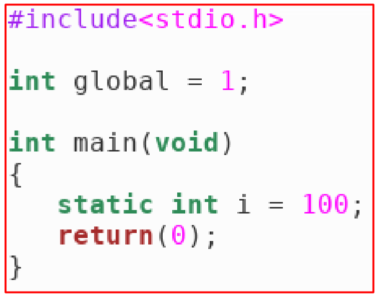
- Note the size of the **initialised** data (bss)



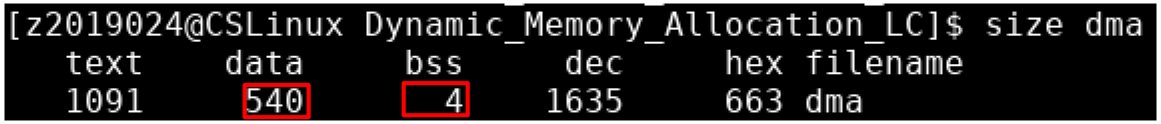
```c
#include<stdio.h>

int global = 1;

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



```
[z2019024@CSLinux Dynamic_Memory_Allocation_LC]$ size dma
   text    data     bss     dec     hex filename
   1091     536       8    1635     663 dma
```



```c
#include<stdio.h>

int global = 1;

int main(void)
{
   static int i = 100;
   return(0);
}
```



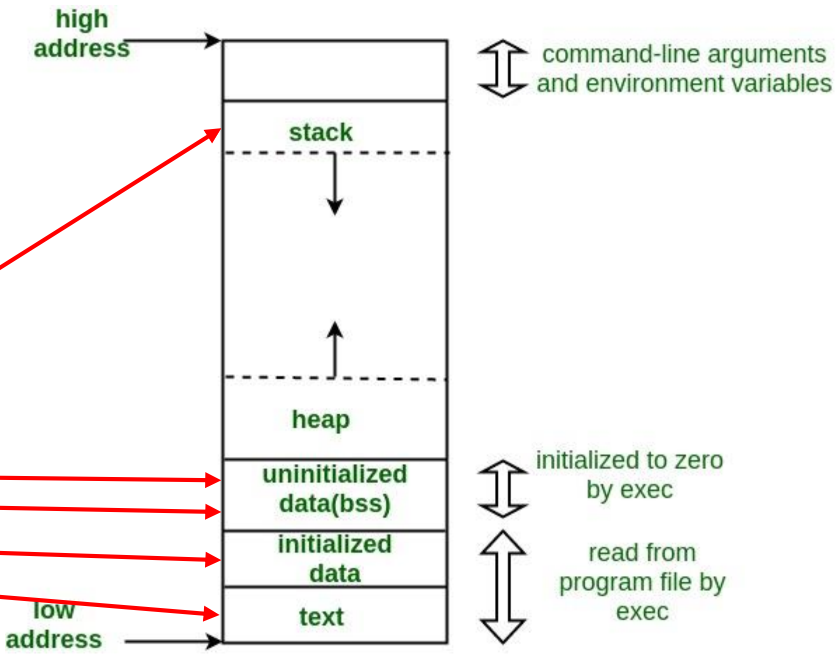
```
[z2019024@CSLinux Dynamic_Memory_Allocation_LC]$ size dma
   text    data     bss     dec     hex filename
   1091     540       4    1635     663 dma
```

Memory Layout of C Programs cont…

- Remember this!
  - The compiler allocates memory (i.e. stack) to store the function's parameters and the variables when the function is called.
  - Once it's terminated, the memory is automatically deallocated.
  - … and **YES**, main is a function!!

Memory Layout of C Programs cont…

```c
#include<stdio.h>

int global;

int main(void)
{
    static int i = 100;
    static int j;

    int k;

    printf("%p\n", &k);
    printf("%p\n", &j);
    printf("%p\n", &global);
    printf("%p\n", &i);
    printf("%p\n", main);
    return(0);
}
```


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

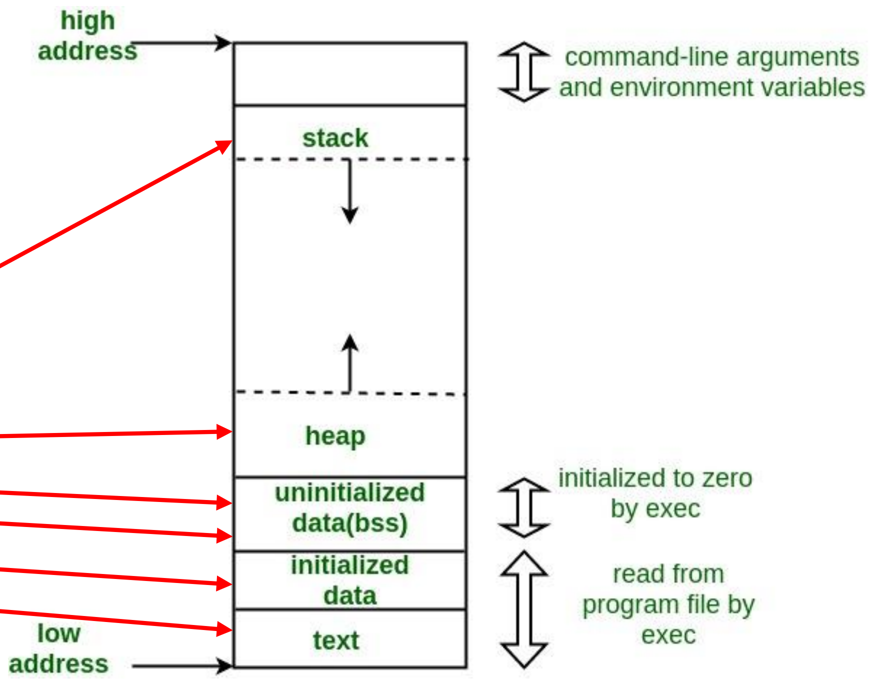
```
[z2019024@CSLinux Dynamic_Memory_Allocation_LC]$ ./dma2
0x7ffca4a7d33c
0x40403c
0x404040
0x404034
0x401132
```

Dynamic Memory Allocation

- Dynamic memory allocation usually takes place in Heap section of memory.
- Memory doesn’t get deallocated at the end of a function call.
- Manage by the programmer using e.g. *malloc* and *free* functions.
- *malloc* and *free* functions are inside *stdlib.h* header file.

Dynamic Memory Allocation cont…

```c
#include<stdio.h>
#include<stdlib.h>

int global;

int main(void)
{
    static int i = 100;
    static int j;

    int k;

    int *p = malloc(sizeof(int));

    printf("%p\n", &k);
    printf("%p\n", &p);
    printf("%p\n", &j);
    printf("%p\n", &global);
    printf("%p\n", &i);
    printf("%p\n", main);
    return(0);
}
```



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

Dynamic Memory Allocation cont…

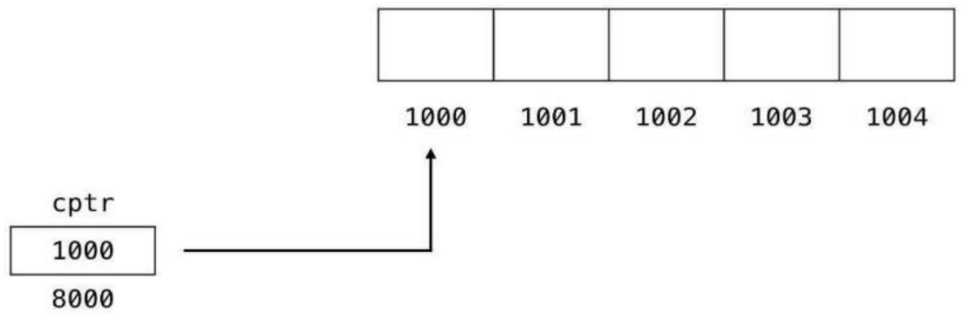
- **malloc()**
  - This function takes parameter as size of memory to be allocated.
  - Returns a pointer to a newly allocated block of memory in the heap.
  - Size is determined in bytes.
  - How to use?

```
int *p = malloc(sizeof(int));
char *q = malloc(sizeof(char));
```

Dynamic Memory Allocation cont…

- Use of Dynamic Memory Allocation:
  - Create dynamic data structures that can change size e.g., lists, trees, graphs.

```
char *cptr = (char *) malloc (5 * sizeof(char));
```



Source: https://www.dyclassroom.com/c/c-dynamic-memory-allocation-malloc-function

Dynamic Memory Allocation cont…

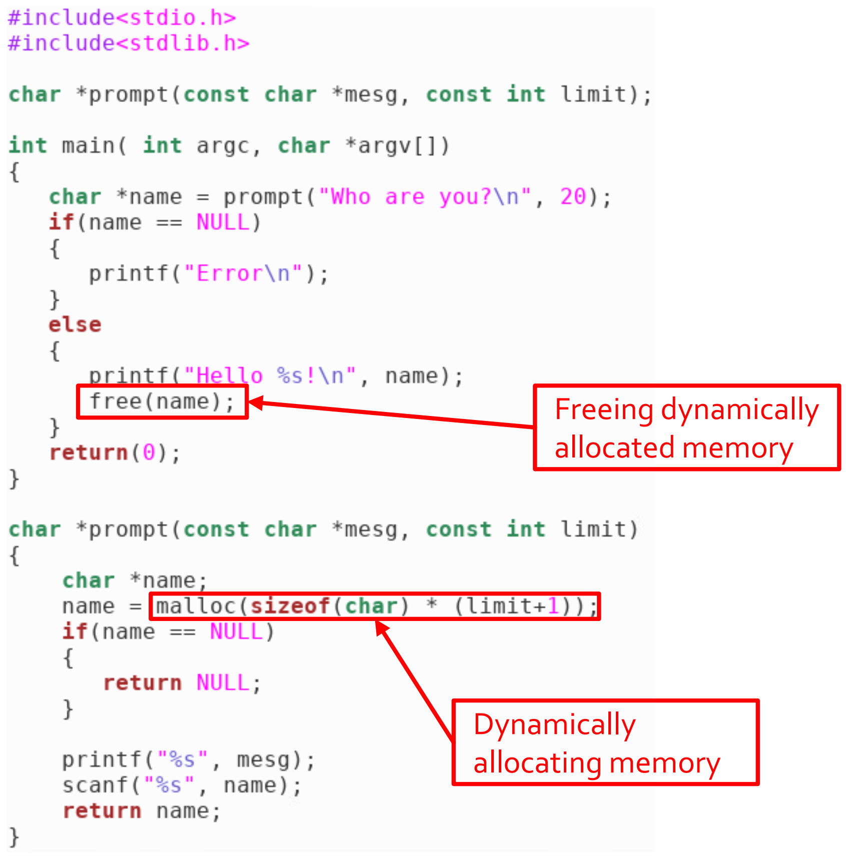
- **free():**
  - To deallocate the block of memory after you have finished using.
  - Trying to free memory not allocated by *malloc* is an error.
  - Trying to free the same memory multiple times is an error.
  - If forget to free memory which no longer required, it can make your program use more and more memory the longer it is running.
  - When the program exits, the OS will reclaim all of the memory, even if it has not been freed.
  - Syntax:

    free(p)

    Where p is the pointer to the memory to be freed.

Dynamic Memory Allocation cont…

- Reusable Prompt: To print a prompt then read in a string.

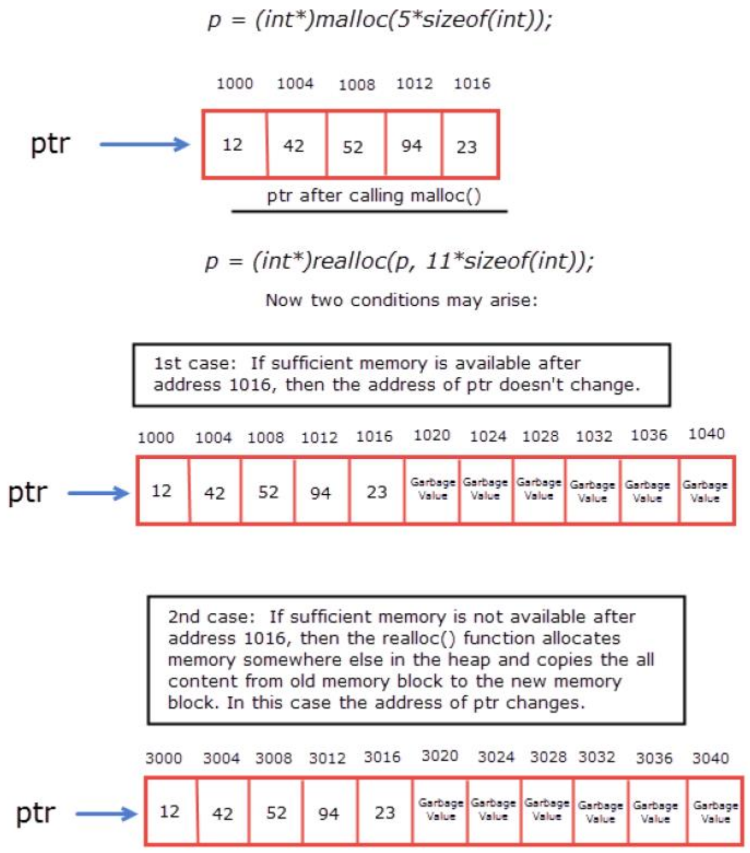


```
#include<stdio.h>
#include<stdlib.h>

char *prompt(const char *mesg, const int limit);

int main( int argc, char *argv[])
{
   char *name = prompt("Who are you?\n", 20);
   if(name == NULL)
   {
      printf("Error\n");
   }
   else
   {
      printf("Hello %s!\n", name);
      free(name);
   }
   return(0);
}

char *prompt(const char *mesg, const int limit)
{
    char *name;
    name = malloc(sizeof(char) * (limit+1));
    if(name == NULL)
    {
       return NULL;
    }

    printf("%s", mesg);
    scanf("%s", name);
    return name;
}
```

Freeing dynamically allocated memory
Dynamically allocating memory

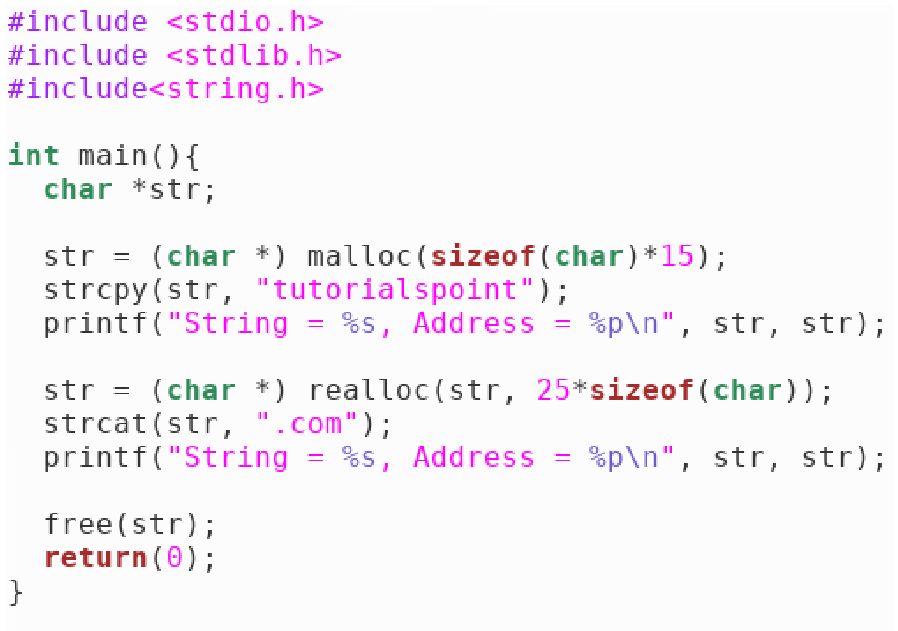
Dynamic Memory Allocation cont...

- **realloc():**
  - To resize the previously allocated memory.

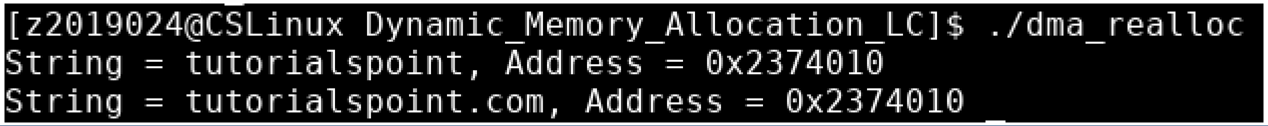

p = (int*)malloc(5*sizeof(int));
1000 1004 1008 1012 1016
ptr
12 42 52 94 23
ptr after calling malloc()
p = (int*)realloc(p, 11*sizeof(int));
Now two conditions may arise:
1st case: If sufficient memory is available after address 1016, then the address of ptr doesn't change.
1000 1004 1008 1012 1016 1020 1024 1028 1032 1036 1040
ptr
12 42 52 94 23 Garbage Value Garbage Value Garbage Value Garbage Value Garbage Value Garbage Value
2nd case: If sufficient memory is not available after address 1016, then the realloc() function allocates memory somewhere else in the heap and copies the all content from old memory block to the new memory block. In this case the address of ptr changes.
3000 3004 3008 3012 3016 3020 3024 3028 3032 3036 3040
ptr
12 42 52 94 23 Garbage Value Garbage Value Garbage Value Garbage Value Garbage Value Garbage Value

Dynamic Memory Allocation cont…

- **realloc()** example



```c
#include <stdio.h>
#include <stdlib.h>
#include<string.h>

int main(){
  char *str;

  str = (char *) malloc(sizeof(char)*15);
  strcpy(str, "tutorialspoint");
  printf("String = %s, Address = %p\n", str, str);

  str = (char *) realloc(str, 25*sizeof(char));
  strcat(str, ".com");
  printf("String = %s, Address = %p\n", str, str);

  free(str);
  return(0);
}
```



```
[z2019024@CSLinux Dynamic_Memory_Allocation_LC]$ ./dma_realloc
String = tutorialspoint, Address = 0x2374010
String = tutorialspoint.com, Address = 0x2374010
```

Conclusion

- Compiler allocated memories are stored in stack segment.
- Dynamically allocated memories are stored in heap segment.
- malloc() is used to dynamically allocate memory.
- free() is used to deallocate dynamically allocated memory.

Thank you!