Dynamic Memory Allocation & Return Pointer from Inside Function

Computer Science Department

Eastern Washington University

Yun Tian (Tony) Ph.D.

Last Class

- Call by reference
- Call by value
- Summary:
  - If we change a **variable itself** inside a function we cannot see the effect outside of the function.
  - If we change **what a variable points to** (if applicable), we could see the effect outside of the function.

Today

- malloc() function → dynamic memory allocation
- Pointer as function parameters
- Return pointer(dynamic array) from inside of function

Dynamic Memory Allocation

- We learned this:
- char name[100]; // I call it **’static array’**
  - size of array is constant
  - we have to know in advance.
  - We call it statically allocated memory. The memory space for **name** array is allocated **automatically and deallocated automatically if inside a function.**

Dynamic Memory Allocation

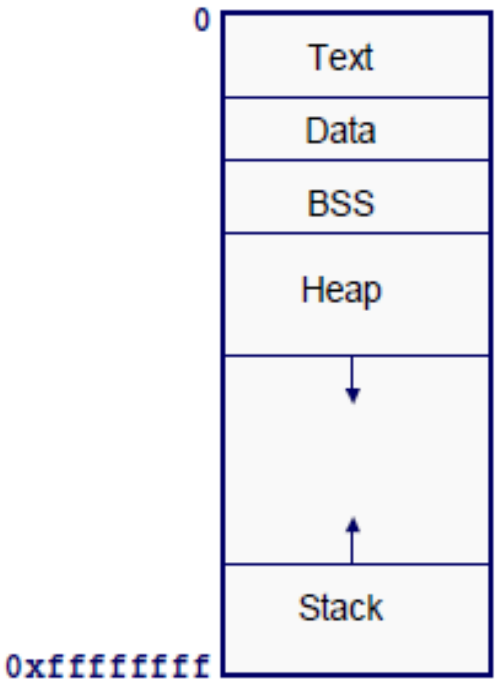
- Consider a situation where you have no idea about the length of the text you need to store.
  - For example you want to store a detailed description about a topic.
  - Here we need to define a pointer to **character** without defining how much memory is required.
  - Later based on requirement we can allocate memory, the requirement either stored in a file or sent by network.

Dynamic Memory Allocation

- Demo memAlloc1.c
- So you have complete control and you can pass any size value while allocating memory.
- Unlike arrays where once you defined the size can not be changed.

Memory Layout

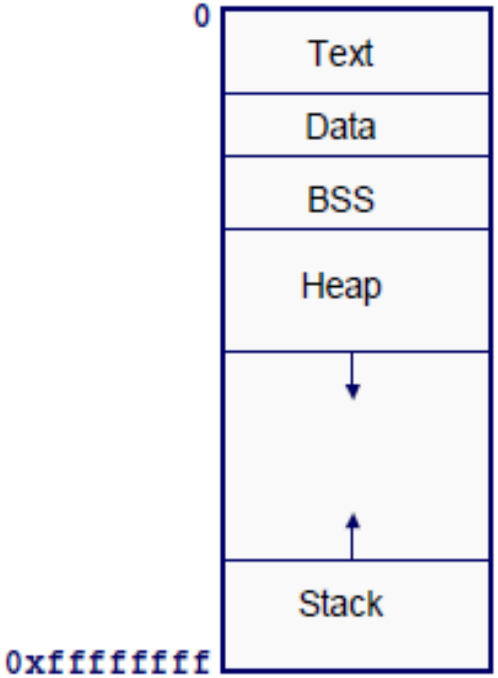
- How is memory organized?
  - Text = code
  - Data = initialized global and static variables
  - BSS = uninitialized global and static variables
  - Stack = **local variables**
  - Heap = **dynamic memory**



Memory Allocation

How is memory allocated?

- Global and static variables → program startup
- Local variables → function call
- Dynamic memory → malloc()



Memory Allocation

```
int iSize;     ← Allocated in BSS, set to zero at main startup
char *f(void)
{
        int i = 10; ←Allocated on stack at start of function f
         char *p; ←Allocated on stack at start of function f
         iSize = 8;
         p = malloc(iSize); ← 8 bytes allocated when call malloc
         return p;       //the memory space p points to
                         // is available outside of this function.


}
```

Memory Deallocation

- How is memory deallocated?
  - Global and static variables → program finish
  - Local variables → function return
  - Dynamic memory → free()
  - All memory is deallocated at program termination.
    - But it is good style to free allocated memory anyway
    - Think about a webserver who has to be running all year around.

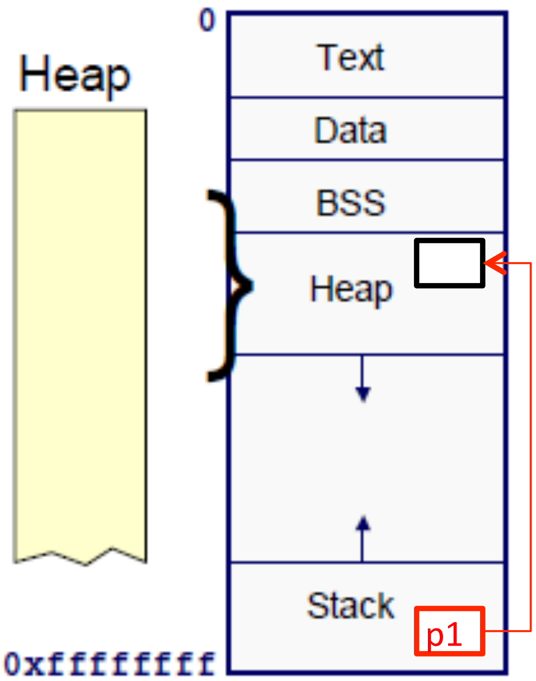
Memory Deallocation

```
int iSize;     ← available until program termination
char *f(void)
{
        int i = 10; ←deallocated by return from function f
        char *p; ←deallocated by return from function f
        iSize = 8;
        p = malloc(iSize);  ← deallocate by calling  free(p)
        return p; //the memory space p points to
                  // is available outside of this function.
}
```

Dynamic Memory Allocation

```
#include <stdlib.h>
void *malloc(size_t size);
void free(void *ptr);
char * f()
{
        char *p1 = malloc(3);
        // 3 bytes memory allocated on the heap,
        //it keeps available until you call free(p1) to deallocate it.
        // Programmers are responsible to free each chunk of memory they have allocated using
        malloc() or calloc().

        char *p2 = malloc(1);
        char *p3 = malloc(4);
        free(p2);
        char *p4 = malloc(6);
        free(p3);
        char *p5 = malloc(2);
        free(p4);
        free(p5);
        return p1;  // we return the address of the black memory box on the heap,
}                    // which is a single value (an address ) returned  to
                     //outside of the function.
                     // The value returned is a copy of the value held by p1.
```



Return Array from Function

- We learned that we cannot return statically allocated array from inside a function.
- The following is **not correct in C**.

```
int [] foo()
{
    int a[10] = {2};
    return a;
}
```

Return Array from Function

- However, you can return a **pointer** from inside a function, which points to a piece of **contiguous** memory locations.
  - We have to use dynamic memory allocation, malloc().
  - Memory allocated is visible outside of function if you return the start address of that piece of memory.
  - In this sense, an initialized pointer is equivalent to a **dynamic array**.
    - That is why **pointer and array name** can be used interchangeably in C.

Return Array from Function

- Demo
  - – returnArr1.c
  - – returnArr2.c

Summary

- Memory Allocation
- Memory Deallocation, how and when
- Memory layout
- Return array from inside of function

Next Class

- Array as function parameters
- Debug tool, gdb