Dynamic Memory Allocation

Last updated 6/22/23

These slides introduce dynamic memory allocation

Dynamic Memory Allocation

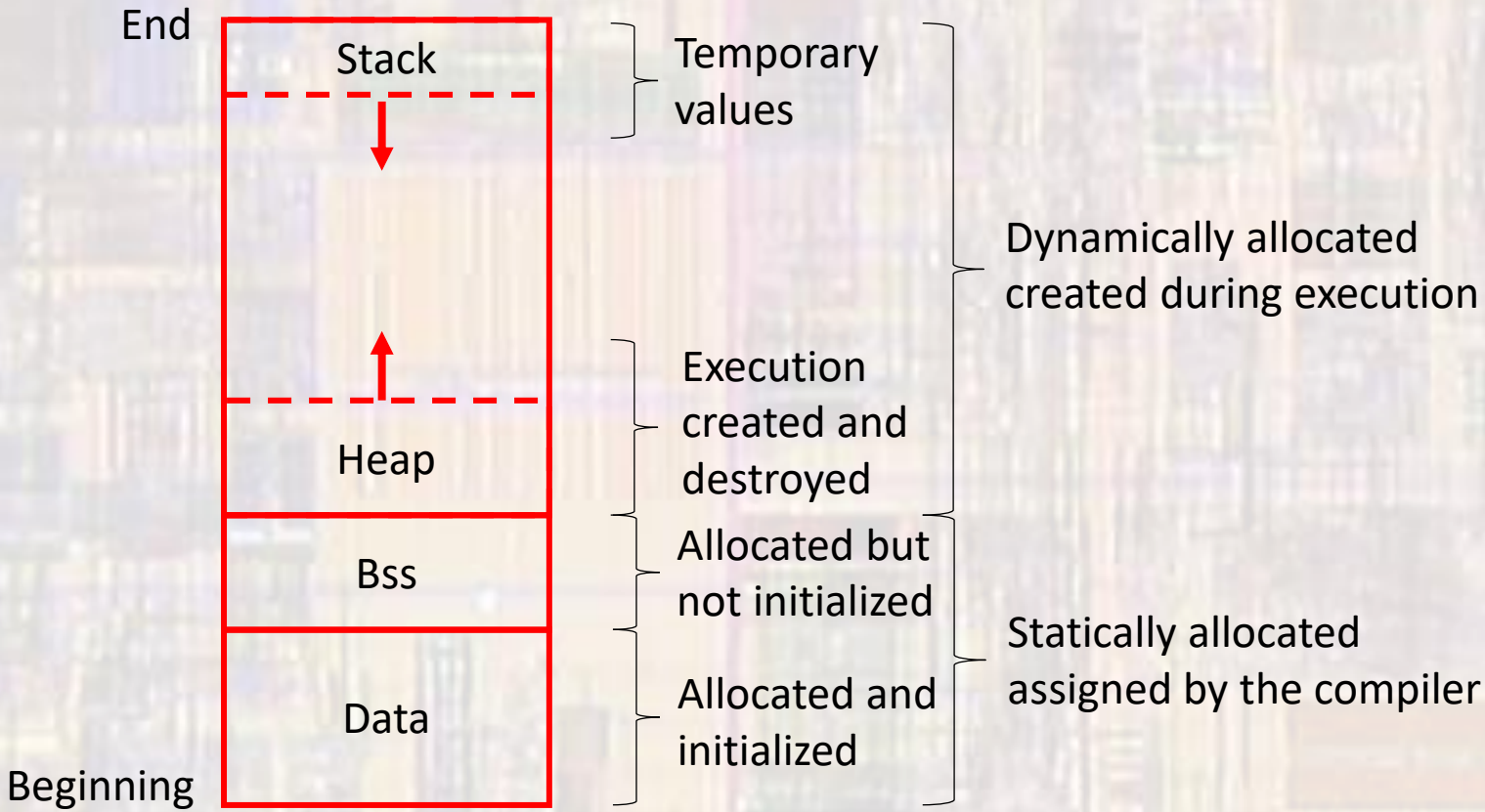
- Stack
  - A section of Data memory
  - Used to hold all temporary variables whose size is known at compile time
    - Return address for a function
    - Copies of parameters passed into a function
    - Return value
    - Temporary variables used in a function
      - Counters, …
      - An array with 26 elements inside a function
      - Note – main is just another function

Dynamic Memory Allocation

- Heap
  - Section of Data memory
  - Dynamic memory
    - Created and destroyed by the program
  - Persists until you de-allocate it
  - Typically, dependent on run time information
    - The heap is used to hold all variables whose size are not known at compile time
      - Store a list of numbers from the user, where the # of inputs is not known ahead of time
  - Can be accessed throughout the program and its functions

Dynamic Memory Allocation

- Data Memory
  - Stack and heap grow towards each other



Dynamic Memory Allocation

- Commands to allocate Heap memory
  - malloc – allocates a block of memory without initialization
  - calloc – allocates a block of memory initialized to 0
  - realloc – changes the total amount of memory allocated

    - All return a pointer – if the memory cannot be allocated (not enough memory left), the pointer is NULL

    - Check to see if the allocation was successful

```
… action to create memory allocation

// check for success
if(mem_ptr == NULL){
   printf("failed to allocate memory");
   exit(0);
}
```

Dynamic Memory Allocation

- Commands
  - malloc – allocates a block of memory without initialization
    - Input parameter is the # of bytes to allocate
    - Returns a void pointer - void pointers can be cast to any other type of pointer
    - Typically use an assignment cast to modify the void pointer

Prototype:

```
void * malloc(size_t size)
```

return type is void * - void pointer

size_t is the size of an integer in the current implementation (think int)

- this is the type returned by sizeof()

size is the number of bytes to allocate

Note: 25 integers would require size to be 100 in a 32b system

Dynamic Memory Allocation

- Example - malloc

```
/*
 * dynamic_mem.c
 *
 *  Created on: Dec 18, 2020
 *      Author: johnsontimoj
 */
//////////////////////////
//
// dynamic memory allocation examples
//
//////////////////////////////
#include <stdio.h>
#include <stdlib.h>

int main(void){
    setbuf(stdout, NULL);  // disable buffering
    int my_ary[10];
    int i;

    // create pointer for memory allocation
    int * mem_ptr;

    // attempt to allocate the memory
    // equivalent for 10 ints
    mem_ptr = malloc(10*sizeof(int));
    if(mem_ptr == NULL){
        printf("failed to allocate memory");
        exit(0);
    }
```

Note the use of sizeof(type) to get the correct # of bytes independent of system

```
    // fill array
    for(i = 0; i < 10; i++){
        my_ary[i] = 4*i;
    }
    // print the array bounds
    printf("%p %p", my_ary, &my_ary[9]);

    // copy to allocated memory
    for(i = 0; i < 10; i++){
        *(mem_ptr + i) = my_ary[i];
    }
    // print memory section + 1 too far
    for(i = 0; i < 11; i++){
        printf("%i\n", *(mem_ptr+i));
    }
    // print the memory bounds
    printf("%p %p", mem_ptr, mem_ptr + 9);
    return 0;
}// end main
```

```
<terminated> (exit value: 0) C
0061FEF0 0061FF140
4
8
12
16
20
24
28
32
36
1997987409
00711DC8 00711DEC
```

Stack memory locations

Heap memory locations

Note garbage outside allocated range

Dynamic Memory Allocation

- Commands
  - calloc – allocates a block of memory initialized to 0
    - Input parameters:
      - # of elements to allocate
      - Size (in bytes) of each element
    - Returns a void pointer - void pointers can be cast to any other type of pointer
    - Typically use an assignment cast to modify the void pointer

Prototype:

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

return type is void * - void pointer

size_t is the size of an integer in the current implementation (think int)
- this is the type returned by sizeof()

nmemb is the # of elements to reserve space for

size is the size of each element in bytes

Dynamic Memory Allocation

- Example - calloc

```
/*
 * dynamic_mem.c
 *
 *  Created on: Dec 18, 2020
 *      Author: johnsontimoj
 */
//////////////////////////
//
// dynamic memory allocation examples
//
////////////////////////////////
#include <stdio.h>
#include <stdlib.h>

int main(void){
    setbuf(stdout, NULL);  // disable buffering
    int i;

    // create pointer for memory allocation
    int * mem_ptr;

    // attempt to allocate the memory
    // equivalent for 10 ints
    mem_ptr = calloc(10, sizeof(int));
    if(mem_ptr == NULL){
        printf("failed to allocate memory");
        exit(0);
    }
```

```
    // verify contents
    // print memory section + 1 too far
    for(i = 0; i < 11; i++){
        printf("%i\n", *(mem_ptr+i));
    }
    // print the memory bounds
    printf("%p %p", mem_ptr, mem_ptr + 9);
    return 0;
}// end main
```

```
<terminated> (exit value: 0) Class_Project.exe [C
0
0
0
0
0
0
0
0
0
0
-499431246
00A51C40 00A51C64
```

Initialized to 0

Note garbage outside allocated range

Dynamic Memory Allocation

- Commands
  - realloc – changes the total amount of memory allocated
    - Input parameters:
      - Pointer to an existing allocated memory block
      - New size of the allocation
    - Returns a void pointer - void pointers can be cast to any other type of pointer
    - Typically use an assignment cast to modify the void pointer

Prototype:

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

return type is void * - void pointer

size_t is the size of an integer in the current implementation (think int)

- this is the type returned by sizeof()

ptr is a pointer to an existing allocated block of memory

size is the new **total** size of the block in bytes

Dynamic Memory Allocation

- Example - realloc

```c
/*
 * dynamic_mem.c
 *
 *  Created on: Dec 18, 2020
 *      Author: johnsontimoj
 */
//////////////////////////
//
// dynamic memory allocation examples
//
//////////////////////////////
#include <stdio.h>
#include <stdlib.h>

int main(void){
    setbuf(stdout, NULL);  // disable buffering
    int i;

    // create pointer for memory allocation
    int * mem_ptr;

    // attempt to allocate the memory
    // equivalent for 10 ints
    mem_ptr = calloc(10, sizeof(int));
    if(mem_ptr == NULL){
        printf("failed to allocate memory");
        exit(0);
    }
```

```c
    // verify contents
    // print memory section + 1 too far
    for(i = 0; i < 11; i++){
        printf("%i\n", *(mem_ptr+i));
    }
    // print the memory bounds
    printf("%p %p\n", mem_ptr, mem_ptr + 9);

    // extend the memory allocation
    // for 5 additional ints
    mem_ptr = realloc(mem_ptr, 15*sizeof(int));
    if(mem_ptr == NULL){
        printf("failed to allocate memory");
        exit(0);
    }

    // verify contents
    // print memory section + 1 too f
    for(i = 0; i < 16; i++){
        printf("%i\n", *(mem_pt
    }
    // print the memory
    printf("%p %p", m
                                 14);

    return 0;
}// end main
```

```
<terminated> (exit value
0
0
0
0
0
0
0
0
0
0
-40091
006                          C64
0
0
0
0
100663302
49808
7085312
7078080
1818582885
808857957
006C1C40 006C1C78
```

Fix this to prove memory is actually allocated – allocate some more

allocated locations cannot be initialized

Dynamic Memory Allocation

- Commands
  - Free – deallocate a block of memory
    - Dynamic memory allocated during program execution persists until either
      - The end of the run
      - The memory is de-allocated
    - Failure to clean up no longer needed allocated memory can cause the program to run out of memory over time
      - Called a memory leak
    - Input parameters:
      - Pointer to an existing allocated memory block

Prototype:

```
void  free(void * ptr)
```

ptr is a pointer to an existing allocated block of memory

All of my examples should have deallocated the Heap memory before ending but we didn't have this command yet