CIS 314

Stack and Heap Structures

Prof. Michel A. Kinsky
Memory Allocation

• There are two types of memory allocation
  ‣ Static memory allocation: Memory is allocated at the start of the program, and freed when program exits
    • Done by the compiler automatically (implicitly)
    • Global variables or objects
      ‣ Alive throughout program execution
      ‣ Can be access anywhere in the program
    • Local variables (inside a function)
      ‣ Memory is allocated when the function starts and freed when the routine returns
      ‣ A local variable cannot be accessed from another function
Memory Allocation

- There are two types of memory allocation
  - Static: Memory is allocated at the start of the program, and freed when program exits

```c
#include <stdio.h>
int number1, number2, number3;
int array[4] = {3, 5, 6, 8};

/* declare and define */
int function (int x){
    int number 4, number5;
    ...
}
void main (void){
    ...
```
Memory Allocation

• There are two types of memory allocation
  ‣ Dynamic memory allocation deals with objects whose size can be adjusted depending on needs
    • Dynamic – Done explicitly by programmer
    • Programmer explicitly requests the system to allocate memory and return starting address of memory allocated
    • This address can be used by the programmer to access the allocated memory
    • When done using memory, it must be explicitly freed
Memory Allocation

- There are two types of memory allocation
  - Dynamic memory allocation deals with objects whose size can be adjusted depending on needs
- Dynamic memory allocation in C:
  - `calloc()`
  - `malloc()`
  - `realloc()`
  - Deallocated using the `free()` function
Memory Allocation

• There are two types of memory allocation
  ‣ Dynamic memory allocation deals with objects whose size can be adjusted depending on needs

```c
#include <stdio.h>
void main (void){
  int i = 0; int nelements_wanted = 8;
  int *i_ptr;
  i_ptr = (int*)malloc(sizeof(int)*nelements_wanted);
  if (i_ptr != NULL) {
    i_ptr[i] = 5;
  }
  else {
    /* Couldn't get the memory - recover */
  }
```
Memory Allocation

• There are two types of memory allocation
  ‣ Dynamic memory allocation deals with objects whose size can be adjusted depending on needs
  ‣ Remember in C if you allocation some piece memory, you are responsible as the programmer to free it
    • x = malloc(n * sizeof(int));
    • /* manipulate x */
    • free(x);
Program memory management

- **Text Segment**
  - Program code
  - Fixed Size

- **Data Segment**
  - Initialized global and static variables
  - Fixed Size

- **BSS Segment**
  - Initialized global and static variables
  - Fixed Size

- **Heap Segment**
  - Dynamic variables managed by malloc(), free(), etc.
  - Variable Size

- **Stack Segment**
  - Stack frames consisting of parameters, return addresses and local variables
  - Variable Size

A stack is a memory area in a computer program that is used by the program to store and retrieve information temporarily. It is a LIFO (Last In First Out) data structure. The bottom of the stack is where new data is added, and the top of the stack is where data is removed. The stack is typically used for function calls, local variables, and parameter passing.

- **Lower addresses**
- **Higher addresses**
- **Bottom of the stack**
- **Top of the stack**
- **Free space**
• Stack and heap are two memory sections in the user mode space
• The stack handles local variables for functions, whose size can be determined at call time
• Some of information saved at function call and restored at function return:
  ‣ Values of callee arguments
  ‣ Register values:
    • Return address (value of PC)
    • Frame pointer (value of FP)
Stack Structure

- Stack will be allocated automatically for function call
- It grows downward to the lower address
- It is Last-in First-out (LIFO) mechanism (tally with the assembly language’s push and pop instructions)
- Even if the stack grows from higher to lower addresses, the local variables on the stack grow from lower to higher addresses
Stack Structure

Stack frame of the function:

```c
int function (int p1, p2, p2){
    int X, Y, Z;
    ...
}
```

Associated C function code:

- Returns address
- Parameter p3
- Parameter p2
- Parameter p1
- Variable X
- Variable Y
- Variable Z

Bottom of stack

Lower addresses

Higher addresses
Heap Structure

• The heap is allocated by demand or request using C memory management functions such as malloc(), memset(), realloc() etc.

• It allows data (especially arrays) to take on variable sizes

• It allows locally created variables to live past end of routine

• This is what permits many structures used in Data Structures and Algorithms
Heap Structure

• It is dynamic allocation, grows upward to the higher memory address

• It is possible to allocate memory and “lose” the pointer to that region without freeing it
  ‣ This is called a memory leak
  ‣ A memory leak can cause the heap to become full

• In a multi-threaded environment each thread will have its own completely independent stack but they will share the heap as needed
Next class

• Compilers & Assembly languages