Dynamic Memory Allocation

Lecture 34

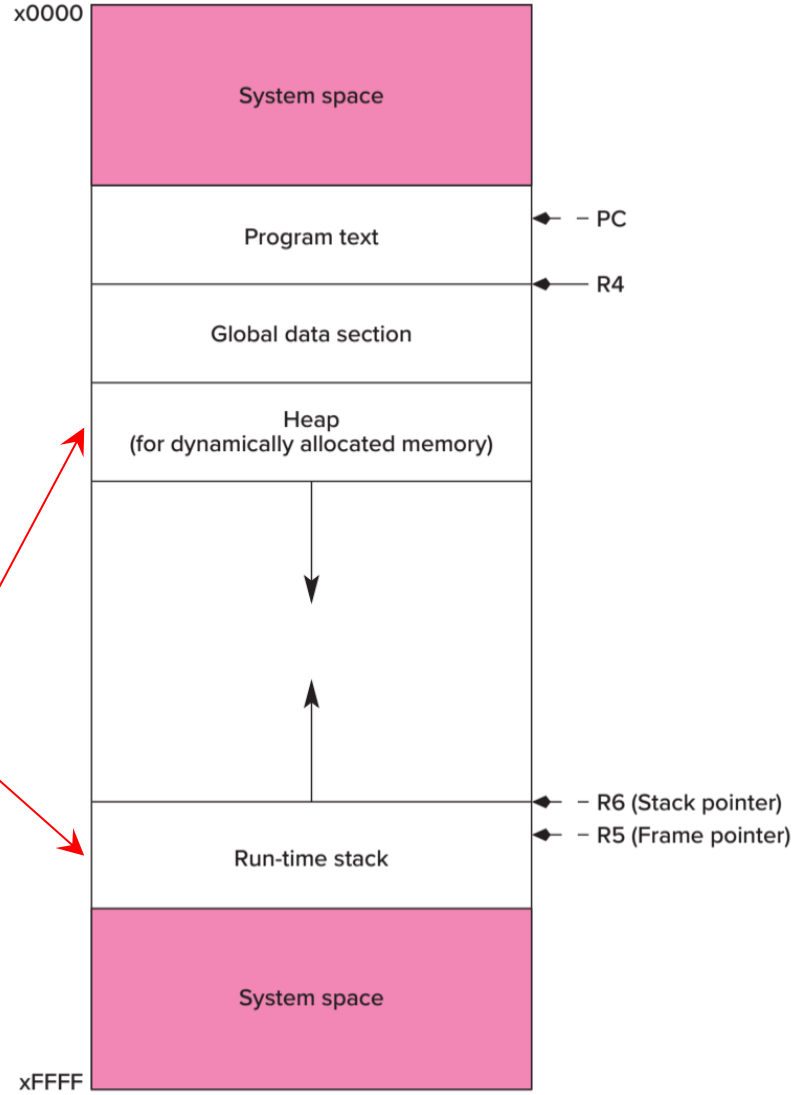
Hyung-Sin Kim

SNU Graduate School of Data Science

Arrays vs. Linked Lists

- Arrays
  - **Pros**: Memory is required only for useful values
  - **Cons**: Hard to precisely size for our needs (redundancy or shortage)
  - C uses **run-time stack** to store arrays
- Linked lists
  - **Pros**: Easy to dynamically add/remove
  - **Cons**: Additional memory is required for linking elements (next/prev pointers)
  - C uses **??** to store linked lists

Heap
Let's see how to use this space!

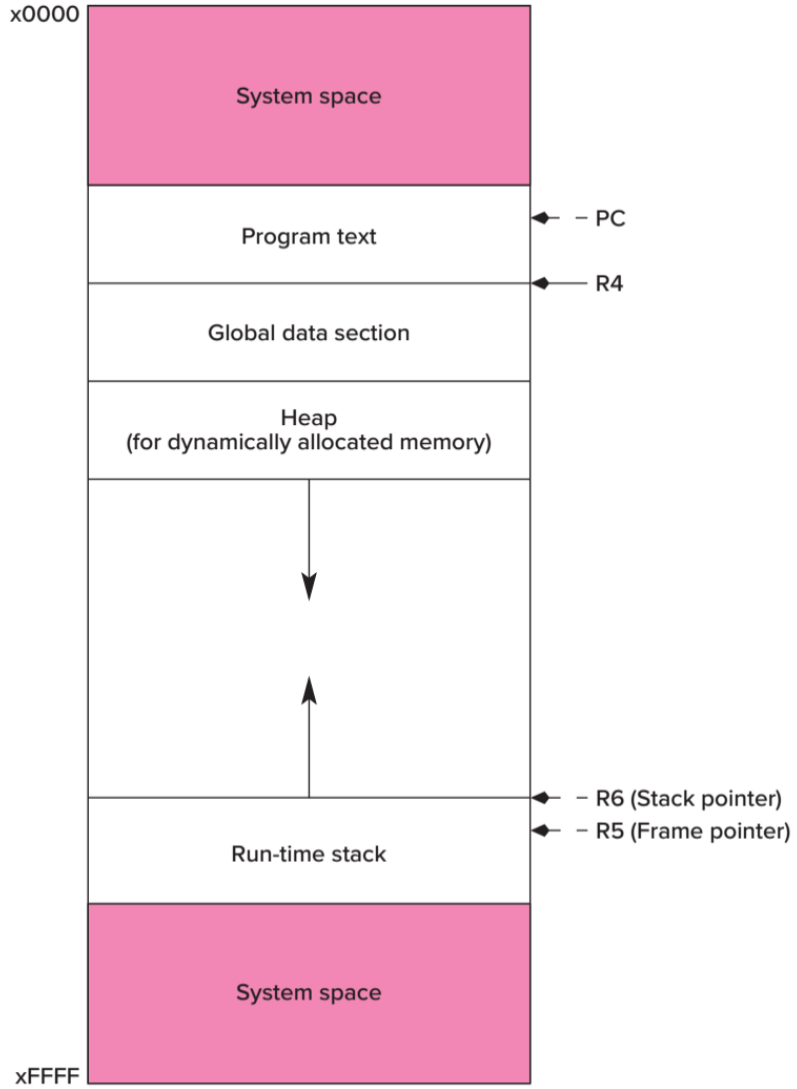


Stack vs. Heap

- Stack
  - Grows upwards
  - Allocation and deallocation are **automatically** done by the compiler
  - Local variable access
  - Faster and no fragmentation
- Heap
  - Grows downwards
  - Allocation and deallocation are **manually** done by the programmer
  - Global variable access
  - Slower and fragmented
- If you are interested more…
  - https://www.guru99.com/stack-vs-heap.html

Dynamic Memory Allocation – malloc

- Allocation: **malloc function**
  - **Parameter**: Requested memory size (bytes)
  - Reserve a contiguous memory block of the requested size in the heap space
  - **Return**: a pointer to the reserved block or NULL when failing to reserve the block (lack of memory)
    - It is safe to check if the return is NULL or not
  - <stdlib.h> needs to be included
- Heap grows downwards as more blocks are allocated
- Once a block is allocated on the heap, the block **survives** until we **explicitly deallocate** it
  - In contrast to variables in stack that are automatically added and removed



Dynamic Memory Allocation – malloc

- Examples
  - int *intPtr;
  - intPtr = malloc(sizeof(int));
  - char *cPtr;
  - cPtr = malloc(sizeof(char));
  - Student *sPtr;
  - sPtr = malloc(STUDENT_NUMS*sizeof(Student));

Hard to expect what return data type users want to get from malloc…

Yes, malloc returns a pointer, but of **what data type**?

A generic pointer of type **void ***
You need to **type cast** the generic pointer for your needs ☺

Dynamic Memory Allocation – malloc

- Examples
  - int *intPtr;
  - intPtr = **(int *)** malloc(sizeof(int));
  - char *cPtr;
  - cPtr = **(char *)** malloc(sizeof(char));
  - Student *sPtr;
  - sPtr = **(Student *)** malloc(STUDENT_NUMS*sizeof(Student));

Type cast!
var = (newType) expression;

Yes, malloc returns a pointer, but of **what data type**?

A generic pointer of type **void ***
You need to **type cast** the generic pointer for your needs ☺

Dynamic Memory Allocation – free

- Examples
  - int *intPtr;
  - intPtr = **(int *)** malloc(sizeof(int));
  - **free(intPtr);**
  - char *cPtr;
  - cPtr = **(char *)** malloc(sizeof(char));
  - **free(cPtr);**
  - Student *sPtr;
  - sPtr = **(Student *)** malloc(STUDENT_NUMS*sizeof(Student));
  - **free(sPtr);**

Once malloc reserves memory on heap, the memory block survives until you deallocate it

Memory deallocation
free(pointer);

A very common error for beginners:
Forgetting deallocation and causing memory overflow in the heap space

Summary

- Dynamic Memory Allocation
  - Arrays vs. Linked Lists
  - Stack vs. Heap
  - malloc

Thanks!