Dynamic Memory Allocation

- Previously ...
  - all memory needs were determined before execution
  - memory is allocated at compile time by the compiler
  - memory is allocated in the stack segment (local variables) or in the data segment (global variables and constants)
  - exact size and type must be known

- What if memory requirements are only known at runtime?
  - memory allocated on-the-fly
  - dynamic memory is allocated in the heap segment
  - exact size and type do not have to be known at compile time
  - we need pointers to save the address of dynamically created space

Operators new and delete

```c
pointer = new type
delete pointer

pointer = new type [size]
delete [] pointer
```
#include <iostream>

int main() {
    int n_elem;
    int *ptr = new int;
    // do something
    *ptr = 20;
    // ...
    delete ptr;

    std::cin >> n_elem;
    int *arr = new int[n_elem];
    // do something
    // ...
    delete [] arr;

    return 0;
}

**Linked Lists**

- Collections of sequential elements stored at **non-contiguous** locations
- Nodes are connected by **links**
  - every nodes keeps a pointer to the next node
- Can **grow** and **shrink** dynamically

**Arrays**

- Can’t make insertions efficiently at both ends
  - insertions in the middle of the array are also expensive
    - what is the cost?

- Can’t make deletions efficiently
  - what is the cost?

### Linked Lists

1. Collections of sequential elements stored at **non-contiguous** locations
2. Nodes are connected by **links**
   - every nodes keeps a pointer to the next node
3. Can **grow** and **shrink** dynamically
Singly Linked List

Operations on Linked Lists

- Linked lists are just collections of sequential data
  - can insert 1 or more elements
    - front, end, by index, by value
  - can delete 1 or more elements
    - front, end, by index, by value
  - can search for a specific element
  - can get an element at a given index
  - can traverse the list
    - visit all nodes and perform an operation (e.g. print or destroy)

Implementing a Singly Linked List

```cpp
class SLList {
private:
  SLNode *head;
  SLNode *tail;
  // all private data/methods
  // ...
public:
  SLList();
  ~SLList();
  // all public methods
  // ...
};
```
class SLNode {
    private:
        int data;
        SLNode *next;
    // all private data/methods
    // ...

    public:
        SLNode(int d);
        ~SLNode();
        // all public methods
        // ...
        friend class SLLList;
};
Delete at front

Delete at end

Delete by value

Delete by index
Doubly Linked List

- Head
- Tail
- NULL pointer

Circular Doubly Linked List

- Head
- Tail