MECHTRON 2MD3

Data Structures and Algorithms for Mechatronics

Winter 2022

09 Elementary Data Structures

Department of Computing and Software

Instructor:

Omid Isfahanialamdari

February 2, 2022

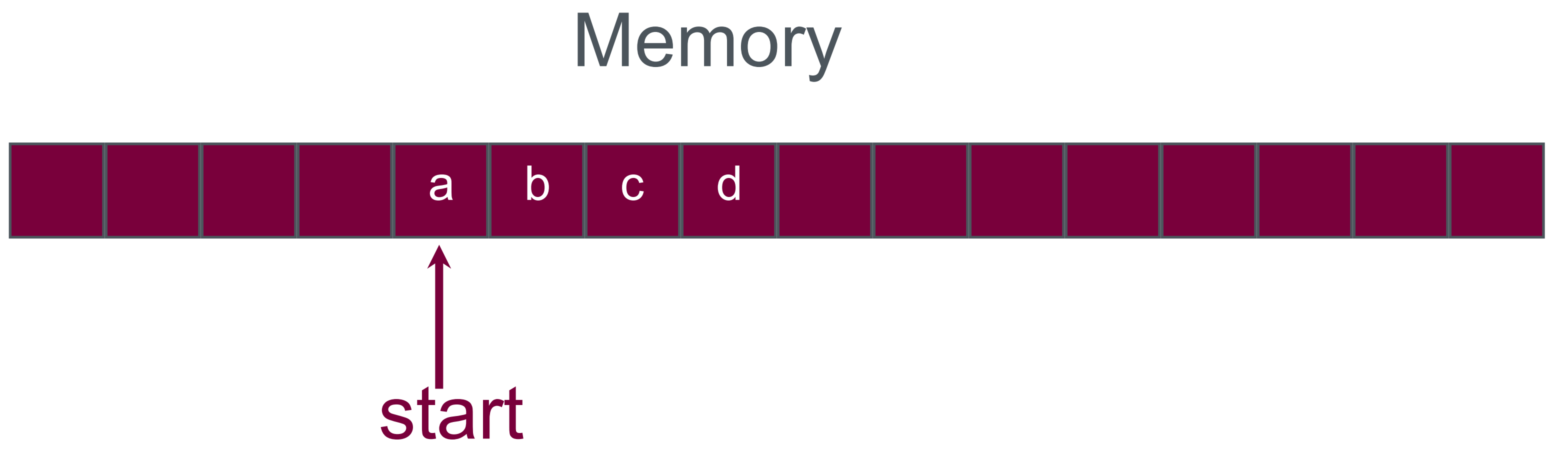
DS and Algorithms

- Algorithms
  - methods for solving problems that are suited for computer implementation.
  - In computer science an algorithm is used to describe a finite, deterministic, and effective problem solving method suitable for implementation as a computer program.

- Data Structure
  - a data structure is a data organization, management, and storage format that enables efficient access and modification.
    - add items
    - remove items
    - access item at a specific location

Arrays

- An array is a data structure consisting of finite number of elements in a specific order, all are of the same type
  - Storing data in a sequential memory locations
  - Access each element using integer index
  - Very basic, popular, and simple
  - Examples: int a[10]; int *a = new int(10);



Example DS using Array

- **Storing high-score game entries**
- A class for storing each game score

```
class GameEntry {                              // a game score entry
public:
  GameEntry(const string& n="", int s=0); // constructor
  string getName() const;                      // get player name
  int getScore() const;                        // get score
private:
  string name;                                 // player's name
  int score;                                   // player's score
};

GameEntry::GameEntry(const string& n, int s) // constructor
  : name(n), score(s) { }
                                               // accessors
string GameEntry::getName() const { return name; }
int GameEntry::getScore() const { return score; }
```

Example DS using Array

- The class that stores high-score game entries

```
class Scores {                                      // stores game high scores
public:
  Scores(int maxEnt = 10);                          // constructor
  ~Scores();                                        // destructor
  void add(const GameEntry& e);                     // add a game entry
  GameEntry remove(int i)                           // remove the ith entry
      throw(IndexOutOfBounds);
private:
  int maxEntries;                                   // maximum number of entries
  int numEntries;                                   // actual number of entries
  GameEntry* entries;                               // array of game entries
};

Scores::Scores(int maxEnt) {                        // constructor
  maxEntries = maxEnt;                              // save the max size
  entries = new GameEntry[maxEntries];              // allocate array storage
  numEntries = 0;                                   // initially no elements
}

Scores::~Scores() {                                 // destructor
  delete[] entries;
}
```

Example DS using Array - Add

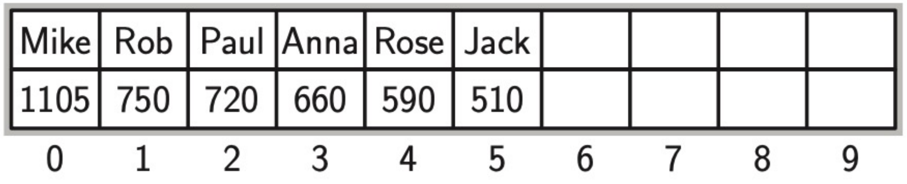
- Only the **highest** maxEntries scores are going to be retained
- add(e):
  - Insert game entry e into the collection of high scores. If this causes the number of entries to exceed maxEntries, the smallest is removed.

```
void Scores::add(const GameEntry& e) {     // add a game entry
  int newScore = e.getScore();             // score to add
  if (numEntries == maxEntries) {          // the array is full
    if (newScore <= entries[maxEntries-1].getScore())
      return;                              // not high enough - ignore
  }
  else numEntries++;                       // if not full, one more entry

  int i = numEntries-2;                    // start with the next to last
  while ( i >= 0 && newScore > entries[i].getScore() ) {
    entries[i+1] = entries[i];             // shift right if smaller
    i--;
  }
  entries[i+1] = e;                        // put e in the empty spot
}
```

Example DS using Array - Add

- Only the **highest** maxEntries scores are going to be retained

```
void Scores::add(const GameEntry& e) {       // add a game entry
  int newScore = e.getScore();               // score to add
  if (numEntries == maxEntries) {            // the array is full
    if (newScore <= entries[maxEntries−1].getScore())
      return;                                // not high enough - ignore
  }
  else numEntries++;                         // if not full, one more entry

  int i = numEntries−2;                      // start with the next to last
  while ( i >= 0 && newScore > entries[i].getScore() ) {
    entries[i+1] = entries[i];               // shift right if smaller
    i−−;
  }
  entries[i+1] = e;                          // put e in the empty spot
}
```

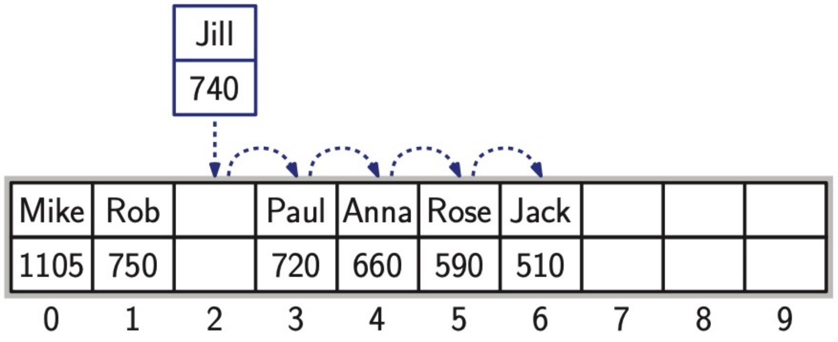


Mike	Rob	Paul	Anna	Rose	Jack				
1105	750	720	660	590	510				
0	1	2	3	4	5	6	7	8	9

Example DS using Array - Add

- Only the **highest** maxEntries scores are going to be retained

```
void Scores::add(const GameEntry& e) {    // add a game entry
  int newScore = e.getScore();            // score to add
  if (numEntries == maxEntries) {         // the array is full
    if (newScore <= entries[maxEntries−1].getScore())
      return;                             // not high enough - ignore
  }
  else numEntries++;                      // if not full, one more entry

  int i = numEntries−2;                   // start with the next to last
  while ( i >= 0 && newScore > entries[i].getScore() ) {
    entries[i+1] = entries[i];            // shift right if smaller
    i−−;
  }
  entries[i+1] = e;                       // put e in the empty spot
}
```



Example DS using Array - Add

- Only the **highest** maxEntries scores are going to be retained

```
void Scores::add(const GameEntry& e) {      // add a game entry
  int newScore = e.getScore();              // score to add
  if (numEntries == maxEntries) {           // the array is full
    if (newScore <= entries[maxEntries−1].getScore())
      return;                               // not high enough - ignore
  }
  else numEntries++;                        // if not full, one more entry

  int i = numEntries−2;                     // start with the next to last
  while ( i >= 0 && newScore > entries[i].getScore() ) {
    entries[i+1] = entries[i];              // shift right if smaller
    i−−;
  }
  entries[i+1] = e;                         // put e in the empty spot
}
```

Mike	Rob	Jill	Paul	Anna	Rose	Jack			
1105	750	740	720	660	590	510			
0	1	2	3	4	5	6	7	8	9

Example DS using Array - Remove

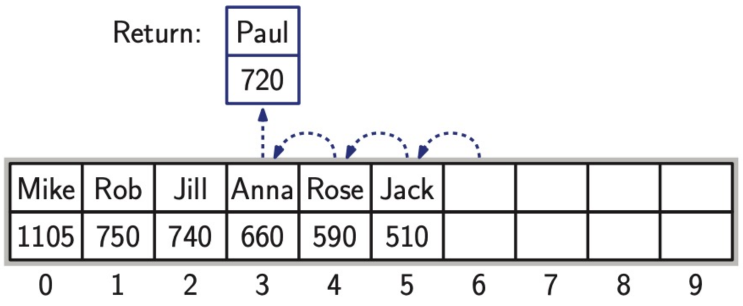
- Only the **highest** maxEntries scores are going to be retained
- remove(i):
  - Remove and return the game entry **e** at index **i** in the **entries** array. If index **i** is outside the bounds of the **entries** array, then this function throws an exception; otherwise, the **entries** array is updated to remove the object at index **i** and all objects previously stored at indices higher than **i** are "shifted left" to fill in for the removed object.

```
GameEntry Scores::remove(int i) throw(IndexOutOfBounds) {
  if ((i < 0) || (i >= numEntries))           // invalid index
    throw IndexOutOfBounds("Invalid index");
  GameEntry e = entries[i];                   // save the removed object
  for (int j = i+1; j < numEntries; j++)
    entries[j−1] = entries[j];                // shift entries left
  numEntries−−;                               // one fewer entry
  return e;                                   // return the removed object
}
```

Example DS using Array - Remove

- Only the **highest** maxEntries scores are going to be retained
- remove(i):
  - Remove and return the game entry **e** at index **i** in the **entries** array. If index **i** is outside the bounds of the **entries** array, then this function throws an exception; otherwise, the **entries** array is updated to remove the object at index **i** and all objects previously stored at indices higher than **i** are "shifted left" to fill in for the removed object.

```
GameEntry Scores::remove(int i) throw(IndexOutOfBounds) {
  if ((i < 0) || (i >= numEntries))          // invalid index
    throw IndexOutOfBounds("Invalid index");
  GameEntry e = entries[i];                    // save the removed object
  for (int j = i+1; j < numEntries; j++)
    entries[j-1] = entries[j];                 // shift entries left
  numEntries--;                                // one fewer entry
  return e;                                    // return the removed object
}
```



Questions?