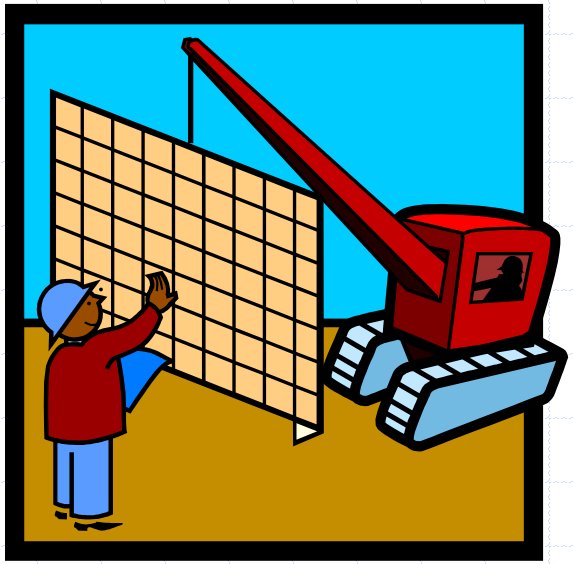
Arrays

Section 3.1



Arrays

- We study some applications of arrays. We start with an array to hold the high scores of a game.
- First we design the class to hold an individual high score.

```
class GameEntry {
public:
   GameEntry(const string& n="", int s=0);
   string getName() const;
   int getScore() const;
private:
   string name;
   int score;
};
```

GameEntry Definitions

- Here are the implementations of the GameEntry member functions, suitable for a .cpp file.

```
GameEntry::GameEntry(const string& n, int s)
    : name(n), score(s) { }

string GameEntry::getName() const { return name; }
int GameEntry::getScore() const { return score; }
```

A Class for High Scores

```
class Scores {
public:
    Scores(int maxEnt = 10);
    ~Scores();
    void add(const GameEntry& e);
    GameEntry remove(int i)
        throw(indexOutOfBoundsException);
private:
    int maxEntries;
    int numEntries;
    GameEntry* entries;
}
```

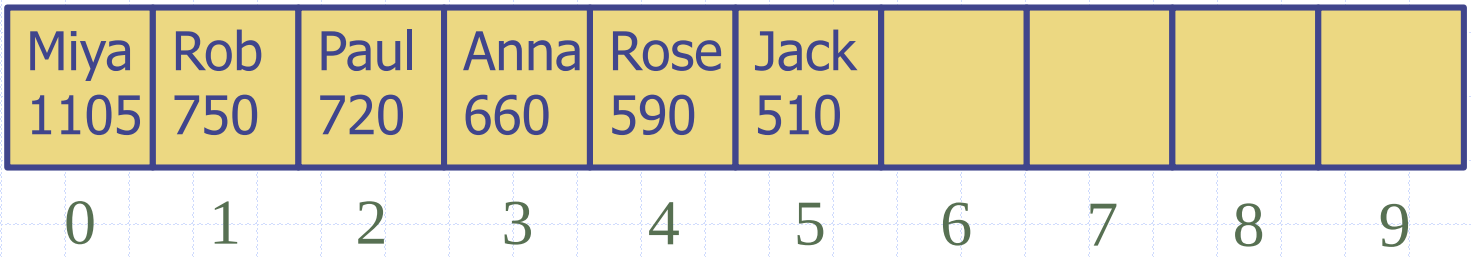
Constructor and Destructor for Scores

```
Scores::Scores(int maxEnt) {
    maxEntries = maxEnt;
    entries = new GameEntry[maxEntries];
    numEntries = 0;
}

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

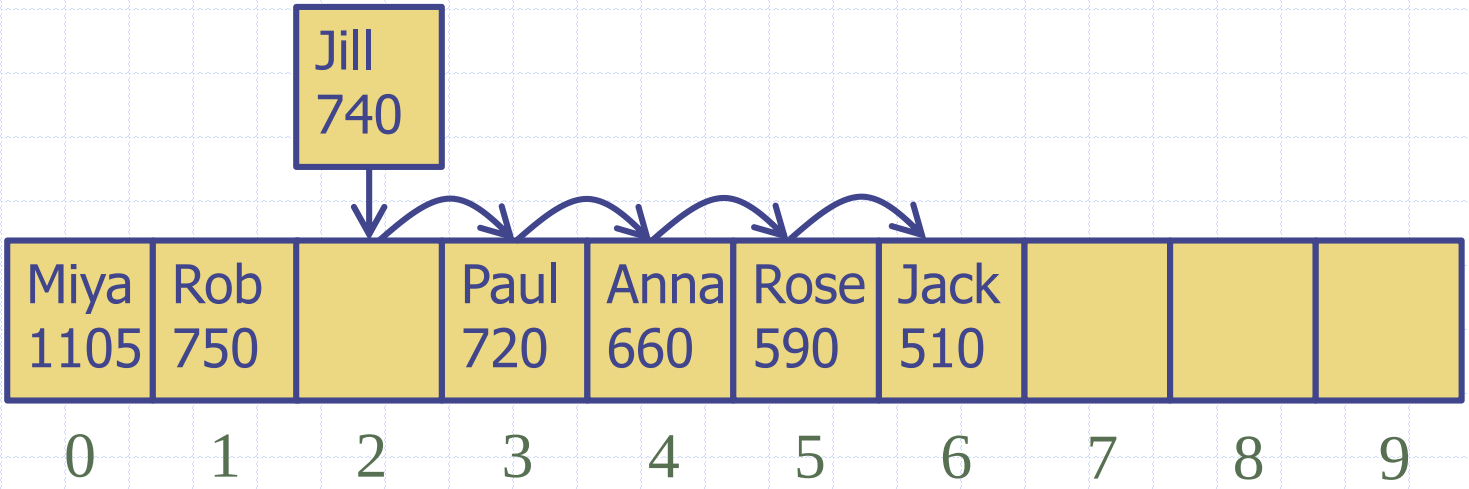
Scores Design

- We choose to keep the high scores sorted from highest to lowest. This is not the only choice we could make.
- Here is an example:



Insertion

- In add(e), we must prepare to insert e by moving all lower scores to the right.



- If we already have the maximum number of scores, then the lowest one is discarded.

Insertion

```
void Scores::add(const GameEntry& e) {
    int newScore = e.getScore();
    if (numEntries == maxEntries) {
        if (newScore <= entries[maxEntries-1].getScore())
            return;
    }
    else numEntries++;

    int i = numEntries - 2;
    while ( i >= 0 && newScore > entries[i].getScore() ) {
        entries[i+1] = entries[i];
        i--;
    }
    entries[i+1] = e;
}
```

Insertion

```
void Scores::add(const GameEntry& e) {
    int newScore = e.getScore();
    if (numEntries == maxEntries) {
        if (newScore <= entries[maxEntries-1].getScore())
            return;
    }
    else numEntries++;

    int i = numEntries - 1;
    while ( i > 0 && newScore > entries[i-1].getScore() ) {
        entries[i] = entries[i-1];
        i--;
    }
    entries[i] = e;
}
```

TIMTOWTDI

- Pronounced "Tim-toady"

There Is More Than One Way To Do It.

- But not all ways are equal. Must check that all limiting cases are handled correctly.

Removal

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 *IndexOutOfBoundsException*. 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.

Similar to add(), but in reverse.

Removal

```
GameEntry Scores::remove(int i)
    throw(IndexOutOfBoundsException) {
    if ((i < 0) || (i >= numEntries))
        throw IndexOutOfBoundsException(
            "Invalid index");

    GameEntry e = entries[i];
    for (int j = i+1; j < numEntries; j++)
        entries[j-1] = entries[j];
    numEntries--;
    return e;
}
```

Sorting an Array

- We've seen that we can add or remove objects at a certain index i in an array while keeping the previous order of the objects intact.
- Now we consider how to rearrange objects of an array that are ordered arbitrarily into ascending order. This is known as sorting.
- We will use an algorithm known as insertion sort.
  - Start with the first element of the array. It's sorted.
  - Step on to the next element of the array, which we'll call the k-th
  - Insert the k-th element into its proper place in the first k-1.
  - Repeat the last two steps until the n-th element has been inserted.

Insertion Sort Pseudocode

Algorithm InsertionSort(A):

Input: An array A of n comparable elements

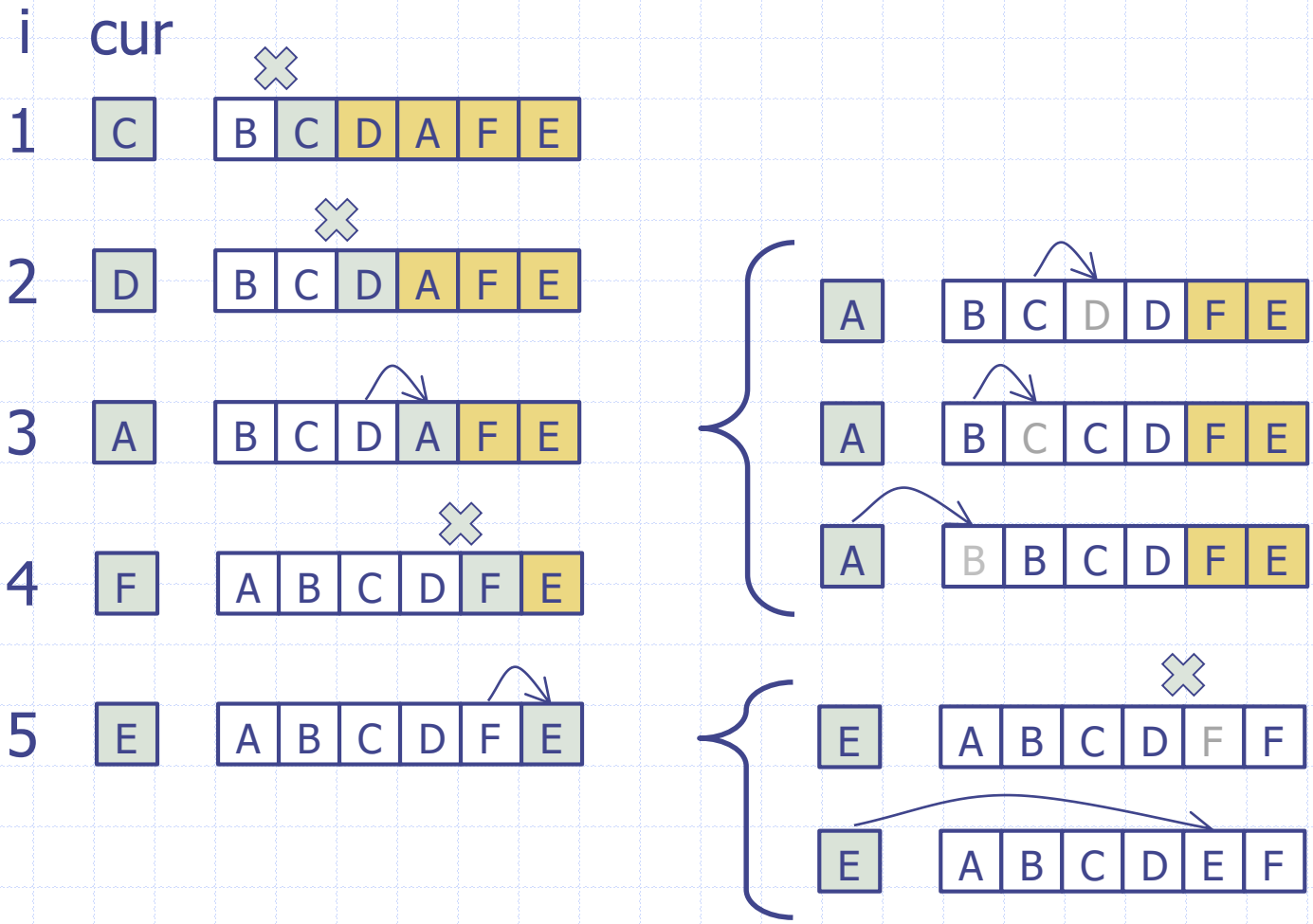
Output: The array A with elements rearranged in nondecreasing order

```
for i ←1 to n-1 do
    {insert A[i] at its proper location in A[0]…A[i-1]}
    cur ← A[i]
    j ← i – 1
    while j ≥0 and A[j] > cur do
        A[j+1] ← A[j]
        j ← j – 1
    A[j+1] ← cur          { cur is now in the right place }
```

Insertion Sort C++

```
void InsertionSort(char* A, int n) {
    for (int i = 1; i < n; i++) {
        char cur = A[i];
        int j = i – 1;
        while ((j >= 0) && (A[j] > cur)) {
            A[j+1] = A[j];
            j--;
        }
        A[j+1] = cur;
    }
}
```

Insertion Sort Example

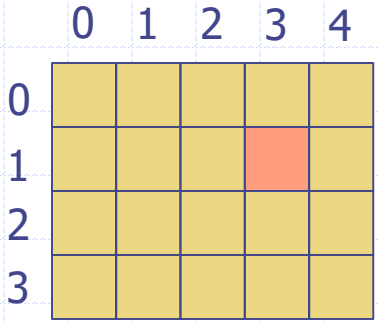


Two-Dimensional Arrays

- We can make arrays that take two indices: a row number and a column number. These are sometimes called matrices (singular: matrix).

```
int M[4][5];
```

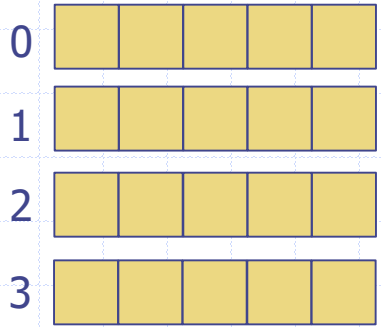
- This example has 4 rows (rows 0, 1, 2, 3) and 5 columns (columns 0, 1, 2, 3, and 4).



M[1][3] is shown in red.

Arrays of Arrays

- A two-dimensional array can be thought of as an array of arrays.



- 4 arrays of 5 elements each. Plus one array of 4 elements, each element a 5-element array.
- C++ uses this row-major order.

Dynamic Allocation of Matrices

- If we do not know the size of the matrix in advance, we must allocate it dynamically.
- C++ does not allow dynamic allocation of multidimensional arrays; it only really understands one-dimensional arrays.
- We use the array of arrays idea.

```
int** M = new int*[n];
for (int i = 0; i < n; i++)
    M[i] = new int[m];
```

```
for (int i = 0; i < n; i++)
    delete[] M[i];
delete[] M;
```

A Quick Note on Style

- It is recommended that you not leave any constant integers in your code except 0 and 1, and possibly -1.
- Consider:

```
int schedule[5][8];
```

- Much better:

```
const int NUM_WEEKDAYS = 5;
const int NUM_WORK_HOURS = 8;
...
int schedule[NUM_WEEKDAYS][NUM_WORK_HOURS];
```

A Quick Note on Style, continued

- This actually applies to any constants, including string constants.

```
Toolkit::use(char* tool) {
    // …
    if(tool == “screwdriver”) {
        // …
    }
}

toolkit->use(“screwdirver”);
```

```
const char* SCREWDRIVER =
    “screwdriver”;

Toolkit::use(char* tool) {
    // …
    if(tool == SCREWDRIVER) {
        // …
    }
}

toolkit->use(SCREWDIRVER);
```