

## Topic: Arrays

- Strings are arrays of *char*
- Declaring arrays
- Arrays of structures and objects
- *Collection* using an object that contains an array
- Array parameters are passed by reference
- Avoiding array boundary violations
- Multi-dimensional arrays
- Vectors

David Keil 2/03 1

## A string is an array of *char*

```

void main(void)
{
  char name[80];
  printf("Your name? ");
  scanf("%s",name);
  printf("Your initial is %c\n",
        name[0]);
}
    
```

[initial.c]

David Keil 2/03 2

## C-style strings use null terminator

- The string library and I/O functions treat the ASCII value 0 as a sentinel that terminates a string

```

char name[6] = "Bill";
    
```

B	i	l	l	\0	
13	10	17	14	0	

- Assigning '\0' to a character element of a string may shorten the string

```

name[3] = '\0';
cout << name;
    
```

Output:

Bi

David Keil 2/03 3

## Concatenating strings

```

void strcat(char destination[],char source[]);
void main()
{
  char title[80] = "Frog ";
  strcat(title,"Prince");
  cout << title << endl;
}
    
```

[strcat.cpp]

```

void strcat(char destination[],char source[])
// Concatenates <source> to <destination>.
{
  int i=0,j=0;
  while (destination[i] != '\0')
    i++;
  do
    destination[i++] = source[j];
  while (source[j++] != '\0');
}
    
```

David Keil 2/03 4

## An array works like a random-access file

[0]	[1]	[2]	[3]	[4]	[5]	[6]	

- Program may instantly store or retrieve data
- Components of array or file are found by calculating their offset from the first one
- We may implement a collection as an array or random file of structures or objects

David Keil 2/03 5

## Building an array

```

#define MAX 3
void main(void)
{
  int income[MAX],
  sum=0,
  i;
  income[0] = 258;
  income[1] = 192;
  income[2] = 467;
  for (i=0; i < MAX; ++i)
  {
    printf("%i ",income[i]);
    sum += income[i];
  }
  printf("\nTotal: %d\n",sum);
}
    
```

[income.c]

Output: 258 192 467 Total: 917

David Keil 2/03 6

## Array initialization

```

void main()
{
    int days_in_month[] =
        {31,28,31,30,31,30,31,31,30,31,30,31};
    cout << "February has " << days_in_month[1]
        << " days" << endl;
}
    
```

*Array of integers*

**Output: February has 28 days**

[month.cpp]

```

void main()
{
    char greeting1[] = "Hello";
    char greeting2[] = {'H','e','l','l','o','\0'};
    cout << greeting1 << " " << greeting2 << endl;
}
    
```

*Two ways to initialize a string*

**Output: Hello Hello**

[strinit.cpp] David Keil 2/03 7

## Calculating variance

```

void main(void)
{
    int score[] = {90,75,84,94,89,97,81},
        i,j,total,avg,
        num_scores = sizeof score / sizeof(int);

    // Find average:
    for (i=0,total=0; i < num_scores; ++i)
        total += score[i];
    avg = total / num_scores;

    // Display variances from average:
    printf("Average = %d\n",avg);
    printf("Score Variance\n");
    for (j=0; j < num_scores; ++j)
        printf("%4d%12d\n",score[j],score[j] - avg);
}
    
```

**Output:**  
 Average = 87  
 Score Variance  
 90 3  
 75 -12  
 84 -3  
 94 7  
 89 2  
 97 10  
 81 -6

*Variance is difference between one item value and average value*

David Keil 2/03 8

## Keyboard input into an array

```

const int MAX_SCORES = 10;
void main()
{
    int score[MAX_SCORES], num_scor = 0, input;
    bool done = false;
    do { // Input:
        cout << "Enter test score (-1 to quit): ";
        cin >> input;
        done = (input < 0 || num_scor >= MAX_SCORES-1);
        if (! done)
        {
            score[num_scor] = input;
            ++num_scor;
        }
    } while (! done);
    for (int i=0; i < num_scor; ++i)
        cout << score[i] << " ";
}
    
```

[See score1.cpp] David Keil 2/03 9

## Appending to an array

```

void main()
{
    // Declare array and variable storing size:
    int A[10] = { 5, 2, 3, 6, 1, 8 };
    int sizeA = 6;

    // Append a 9 to the array:
    A[sizeA] = 9;
    sizeA++;

    // Display result:
    for (int j=0; j < sizeA; ++j)
        cout << A[j] << " ";
    cout << endl;
}
    
```

**Output: 5 2 3 6 1 8 9**

[append.cpp] David Keil 2/03 10

## Inserting at start of an array

```

void main()
{
    // Declare array and var storing its size:
    int A[10] = { 5, 2, 3, 6, 1, 8 };
    int sizeA = 6;

    // Insert a 9 into the array at beginning:
    for (int i=sizeA; i > 0; --i)
        A[i] = A[i-1];
    A[0] = 9;
    sizeA++;

    // Display result:
    for (int j=0; j < sizeA; ++j)
        cout << A[j] << " ";
    cout << endl;
}
    
```

**Output: 9 5 2 3 6 1 8**

[insert.cpp] David Keil 2/03 11

## Reading file into array

```

#include <iostream.h>
#include <fstream.h>

void main()
{
    ifstream fin("file_arr.txt",ios::nocreate);
    if (fin.good())
    {
        int A[200],num_read = 0;
        while (fin >> A[num_read])
            num_read++;
        fin.close();
        for (int i = 0; i < num_read; ++i)
            cout << A[i] << " ";
    }
    else cout << "file_arr.txt not found" << endl;
}
    
```

[file\_arr.cpp] David Keil 2/03 12

### File input to array via function

```

class integer                                     [readarry.cpp]
{
public:
    integer() { value = 0; }
    integer(int n) { value = n; }
    void display() { cout << value << " "; }
    bool retrieve(istream& fin)
        { return (fin >> value) ? true : false; }
private:
    int value;
};

void main()
{
    int size = 0;
    ifstream fin("readarry.txt",ios::nocreate);
    integer A[100];
    while (A[size].retrieve(fin))    size++;
    for (int i=0; i < size; ++i)    A[i].display();
    fin.close();
}
    
```

**Output: 3 6 8 2 5**

David Keil 2/03 13

### Using enumerated-type constants with subscripts

```

enum seasons
{ SPRING, SUMMER, FALL, WINTER };

const char* season_name[] =
    {"Spring", "Summer", "Fall", "Winter"};

void main()
{
    for (int i = SPRING; i <= WINTER; i++)
        cout << season_name[i] << " ";
}
    
```

**Output: Spring Summer Fall Winter**

[season.cpp]  
David Keil 2/03 14

### Array parameters are passed by reference

```

#include <ctype.h>
void capitalize(char s[]);
void main()
{
    char name[] = "bob";
    capitalize(name);
    cout << name << endl;
}

void capitalize(char s[])
// Modifies element 0 of actual parameter
{
    s[0] = toupper(s[0]);
}
    
```

**Output: Bob**

[capitaliz.cpp]  
David Keil 2/03 15

### An int-array parameter

```

void update_element(int A[],int index,int new_val);
void main()
{
    int score[] = {92,86,75,30,92,89};
    for (int i=0; i < sizeof(score) / sizeof(int); ++i)
        cout << score[i] << " ";
    cout << endl;
    update_element(score,3,100);
    for (int i=0; i < sizeof(score) / sizeof(int); ++i)
        cout << score[i] << " ";
}

void update_element(int A[],int index,int new_val)
// Assigns new_val to element of <A> denoted by <index>.
{
    if (index >= 0)
        A[index] = new_val;
}
    
```

**Output: 92 86 75 30 92 89  
92 86 75 100 92 89**

[score.cpp]  
David Keil 2/03 16

### Observing array boundaries

- int n[5];** allocates five ints

<b>[0]</b>	<b>[1]</b>	<b>[2]</b>	<b>[3]</b>	<b>[4]</b>

- Semantically valid subscripts here: 0 to 4
- n[5]** is valid syntax, but refers to memory that is outside the bounds of the declared array

David Keil 2/03 17

### A two-dimensional array

	0	1	2	
← High	3552.2	3560.0	3540.0	
← Low	3539.2	3544.8	3530.6	
← Close	3550.5	3557.7	3530.6	

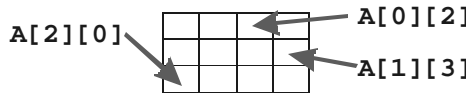
```

const int MAX_DAYS = 30;
enum PRICE_CATEGORY {HIGH,LOW,CLOSE};
double daily_average[CLOSE+1][MAX_DAYS];
    
```

[P. H. Winston, *On to C*, p. 88]  
David Keil 2/03 18

#### Rows and columns in 2D array

- First subscript represents *row* (up-down)



- Second subscript represents *column* (across)
- This is called *row-major ordering*

David Keil 2/03 19

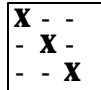
#### Initializing a 2D array

```
const int NUM_DAYS = 5;
enum CATEGORIES {HIGH,LOW,CLOSE};
const char* CATEGORY_NAME[CLOSE+1] =
    {"HIGH","LOW","CLOSE"};
void main()
{
    double price[CLOSE+1][NUM_DAYS] =
        {
            {76.2,81.3,78.5,79.2,80.7},
            {70.9,75.4,71.3,71.8,74.1},
            {74.0,73.6,78.2,76.6,79.5}
        };
    ...
}
```

David Keil 2/03 20

#### Example: Tic-tac-toe board

```
char board[3][3] = {'-','-'};
board[0][0] = 'X';
board[1][1] = 'X';
board[2][2] = 'X';
for(int row = 0; row < 3; ++row)
{
    for(int col = 0; col < 3; ++col)
        cout << board[row][col];
    cout << endl;
}
```



David Keil 2/03 21

#### Summarizing stock averages

```
...
int day, categ;
double categ_total[CLOSE+1] = {0,0,0};
printf("%12s %6s %6s %6s %6s %8s\n",
    "M","Tu","W","Th","F","Average");
for (categ=HIGH; categ <= CLOSE; ++categ)
{
    printf("%-6s",CATEGORY_NAME[categ]);
    for (day = 0; day < NUM_DAYS; ++day)
    {
        printf("%6.1f ",price[categ][day]);
        categ_total[categ] += price[categ][day];
    }
    printf("%8.1f\n",categ_total[categ] / NUM_DAYS);
}
}
Output:
M Tu W Th F Average
HIGH 76.2 81.3 78.5 79.2 80.7 79.2
LOW 70.9 75.4 71.3 71.8 74.1 72.7
CLOSE 74.0 73.6 78.2 76.6 79.5 76.4
```

David Keil 2/03 22

#### String array I: 2D array of char

```
// array of 100 strings,
// 80 characters each
char s[100][80];

// Read words from file "x.txt"
// into array 's':
ifstream fin("x.txt", ios::nocreate);
for (int i=0; ! fin.eof(); i++)
{
    fin >> s[i];
}
fin.close();
```

David Keil 2/03 23

#### String array II: array of char pointers

```
char *s[100];

// Read, copy into dynamically allocated
// string variables:
ifstream fin("x.txt", ios::nocreate);
for (int i=0; ! fin.eof(); i++)
{
    char buf[200];
    fin >> buf;
    s[i] = new char[strlen(buf)+1];
    strcpy(s[i], buf);
}
fin.close();
```

David Keil 2/03 24

## Collection of strings

```

class word_colx
{
public:
    word_colx();
    insert(char *s);
    display();
    read_file(char *file_name);
private:
    char *w[100];
    int num_words;
};
    
```

- Logic on previous slide could be used to implement `read_file` above

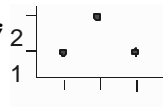
David Keil 2/03 25

## An array of structures

```

struct points { int x,y; };
void point_display(points p);

void main()
{
    points triangle[3] = {{1,1},{2,2},{3,1}};
    for (int i=0; i < 3; ++i)
        point_display(triangle[i]);
}
    
```



```

void point_display(points p)
{
    cout << "(" << p.x << "," << p.y << "),"";
}
    
```

Output: (1,1), (2,2), (3,1)

David Keil 2/03 26

## A class with a 2D-array member

```

class summaries
{
public:
    enum { NUM_DAYS = 5 };
    enum CATEGORIES {HIGH,LOW,CLOSE};
    summaries() {
        for (int i=HIGH; i <= CLOSE; i++)
            for (int j=0; j < NUM_DAYS; j++)
                price[i][j] = 0.0; }
    void display();
private:
    double price[CLOSE+1][NUM_DAYS];
};
    
```

[stockavg.cpp]

*Data is stored in two-dimensional array*

Output:

	M	Tu	W	Th	F	Average
HIGH	0	0	0	0	0	0
LOW	0	0	0	0	0	0
CLOSE	0	0	0	0	0	0

David Keil 2/03 27

## Nested loops to display 2D array

```

void summaries::display()
{
    const char* CATEGORY_NAME[CLOSE+1] = {"HIGH","LOW","CLOSE"};
    double categ_total[CLOSE+1] = {0,0,0};
    cout << " M Tu W Th F Average" << endl;
    for (int categ=HIGH; categ <= CLOSE; ++categ) {
        cout << setw(6) << CATEGORY_NAME[categ];
        for (int day = 0; day < NUM_DAYS; ++day) {
            cout << setiosflags(ios::showpoint || ios::fixed)
                << setw(5) << setprecision(1)
                << price[categ][day];
            categ_total[categ] += price[categ][day];
        }
        cout << setw(6) << categ_total[categ] / NUM_DAYS
            << endl;
    }
}
    
```

David Keil 2/03 28

## A collection stores many items of 1 type

```

class score_lists
{
public:
    enum {MAX_SCORES = 6};
    score_lists() { size = 0; };
    bool append(int new_elt);
    void input();
    void display();
private:
    int element[MAX_SCORES];
    int size;
};
    
```

[score2.cpp]

An instance of score Lists:

5

2 45 9 4 32

*A collection of int*

```

void score_lists::input()
// Keyboard-input member function.
{
    int input_value;
    do {
        cout << "Score (-1 to quit): ";
        cin >> input_value;
    } while (input_value >= 0 &&
        append(input_value));
}
    
```

David Keil 2/03 29

## Managing a collection

```

bool score_lists::append(int new_elt)
// Adds <new_elt> to collection.
{
    bool success = (size < MAX_SCORES);
    if (success) element[size++] = new_elt;
    else
        cout << "score_lists::append: "
            << "collection already full" << endl;
    return success;
}

void score_lists::display()
// Displays values.
{
    cout << endl;
    for (int i = 0; i < size; ++i)
        cout << element[i] << endl;
}
    
```

*Array boundary check to see if there's room*

David Keil 2/03 30

#### A collection of employees

```

class rosters
{
public:
    rosters() { numrecs = 0; }
private:
    employees emp[100];
    int numrecs;
};
    
```

A collection class  
[payroll.cpp]

```

struct employees
{
    char name[40];
    int salary;
};
    
```

An instance of collection class rosters

num\_rec 4 emp [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]

David Keil 2/03 31

#### A relation associates set elements

- An array of structures implements a *relation*
- A relation represents an *association* between elements of two or more sets
- Examples:
  - product names and prices (price list)
  - teams, opponent teams, and dates (game schedule)
  - swimmers and elapsed times
- The term “relation” is from set theory and database theory, not C or C++ terminology

David Keil 2/03 32

#### A location on a coordinate axis

```

class points
{
public:
    points()
    { x = y = 0; }
    points(int x_init, int y_init)
    { x = x_init; y = y_init; }
    bool input(istream& in)
    { return !(in >> x >> y); }
    void display()
    { cout << "(" << x << ", " << y << " ) "; }
private:
    int x, y;
};
    
```

[point.cpp]

David Keil 2/03 33

#### A collection of (x,y) objects

```

class point_collections
{
public:
    point_collections() { numpoints = 0; }
    void retrieve(char* file_name); // defined elsewhere
    void display(); // defined elsewhere
    enum{ MAX_POINTS = 100 };
private:
    points point[MAX_POINTS];
    int numpoints;
};

void main()
{
    point_collections list;
    list.retrieve("point.txt");
    list.display();
}
    
```

David Keil 2/03 34

#### Vectors are expandable containers

- vector* is a predefined class generator (template) in the ANSI/ISO Standard Template Library (STL)
- To use: **#include <vector>**
- You may declare vectors of any element type:  
**vector<int> score;**  
**vector<keilwidget> mywidgets;**
- You may insert values into a vector w/o limit:  
**score.push\_back(95);**
- Vectors have an array-like interface:  
**cout << score[2];**

David Keil 2/03 35

#### Vector example

```

#include "vectdemo.h"

void main()
{
    int A[] = { 1, 3, 4, 2, 7, 5 };
    int Asize = sizeof(A) / sizeof(int);
    vector<int> vec1(A, Asize);
    vec1.push_back(6);
    vector<int> vec2 = vec1;
    cout << "Your vector is "
         << vec2 << endl;
}
    
```

[vectdemo.cpp]

Output: Your vector is 1 3 4 2 7 5 6

David Keil 2/03 36

### A vector of employees

```

employees emp;
emp. input();
vector<employees> roster;
roster.push_back(emp);
cout << roster;
    
```

*Type employees must have been defined, with input, output operations*

*These operations are predefined for vectors*

David Keil 2/03 37

### Communicating collection data between devices and memory

- Hints for homework problem:
- Code example: *point.cpp*

```

void main()
{
    // 1. Input from user to array
    // 2. Counted loop to save to disk
    // 3. Sentinel loop to retrieve
    // 4. Counted loop to display
}
    
```

- A menu system is appropriate to permit user to drive operations

David Keil 2/03 38

### Code examples related to arrays

<b>int A[10];</b>	Array declaration
<b>cout &lt;&lt; A[10];</b>	Array-element output
<b>void f(int A[1]);</b>	Prototype of function with integer-array parameter
<b>void g(int x);</b>	Prototype of function with <i>int</i> parameter
<b>int A[10];</b>	Function call with array parameter
<b>f(A);</b>	Function call with parameter that is an array element
<b>g(A[4]);</b>	Function call with parameter that is an array element
<b>cout &lt;&lt; A;</b>	Syntax error if <i>A</i> is array other than string

David Keil 2/03 39

### Array problems

- Given a set of tasks, each with start and finish times, find the smallest number of non-overlapping tasks that fill the day, from 1:00 to 6:00. *Example:* { (A,1,3), (B,2,4), (C,3,5), (D,4,6), (E, 5,6), (F,1,4), (G,1,2) }
- Given user inputs of a set of 4 quarterly sales figures for each of 3 regions, calculate totals by quarter and by region. *Example:*

	Q1	Q2	Q3	Q4
<b>R1</b>	236	311	398	892
<b>R2</b>	128	232	109	673
<b>R3</b>	776	897	349	1031

David Keil 2/03 40