L E S S O N S E T

7 Arrays

|  |  |  |
| --- | --- | --- |
| PURPOSE | 1. | To introduce and allow students to work with arrays |
|  | 2. | To introduce the **typedef** statement |
|  | 3. | To work with and manipulate multidimensional arrays |
| PROCEDURE | 1. | Students should read the Pre-lab Reading Assignment before coming to lab. |
|  | 2. | Students should complete the Pre-lab Writing Assignment before coming to lab. |
|  | 3. | In the lab, students should complete labs assigned to them by the instructor. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Contents | Pre-requisites | Approximate completion time | Page number | Check when done |
| Pre-lab Reading Assignment |  | 20 min. | 114 |  |
| Pre-lab Writing Assignment | Pre-lab reading | 10 min. | 122 |  |
| LESSON 7A |  |  |  |  |
| Lab 7.1  Working with One- | Basic understanding of | 30 min. | 123 |  |
| Dimensional Arrays | one-dimensional arrays |  |  |  |
| Lab 7.2  Strings as Arrays of | Basic understanding of | 20 min. | 126 |  |
| Characters | arrays of characters |  |  |  |
| LESSON 7B |  |  |  |  |
| Lab 7.3  Working with Two- | Understanding of multi- | 30 min. | 129 |  |
| Dimensional Arrays | dimensional arrays |  |  |  |
| Lab 7.4  Student Generated Code | Basic understanding | 30 min. | 134 |  |
| Assignments | of arrays |  |  |  |

113

# P R E - LA B R E A D I N G A S S I G N M E N T

## One-Dimensional Arrays

So far we have talked about a variable as a single location in the computer’s memory. It is possible to have a collection of memory locations, all of which have the same data type, grouped together under one name. Such a collection is called an **array**. Like every variable, an array must be defined so that the com- puter can “reserve” the appropriate amount of memory. This amount is based upon the type of data to be stored and the number of locations, i.e., size of the array, each of which is given in the definition.

*Example:* Given a list of ages (from a file or input from the keyboard), find and display the number of people for each age.

The programmer does not know the ages to be read but needs a space for the total number of occurrences of each “legitimate age.” Assuming that ages 1, 2, . . . , 100 are possible, the following array definition can be used.

const int TOTALYEARS = 100;

int main()

{

**int ageFrequency[TOTALYEARS];** //reserves memory for 100 ints

:

return 0;

}

Following the rules of variable definition, the data type (integer in this case) is given first, followed by the name of the array (ageFrequency), and then the total number of memory locations enclosed in brackets. The number of memory loca- tions must be an integer expression greater than zero and can be given either as a named constant (as shown in the above example) or as a literal constant (an actual number such as 100).

Each element of an array, consisting of a particular memory location within the group, is accessed by giving the name of the array and a position with the array (subscript). In C++ the subscript, sometimes referred to as index, is enclosed in square brackets. The numbering of the subscripts always begins at 0 and ends with one less than the total number of locations. Thus the elements in the ageFrequency array defined above are referenced as ageFrequency[0] through ageFrequency[99].

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

0 1 2 3 4 5 . . . . . . 97 98 99

If in our example we want ages from 1 to 100, the number of occurrences of age 4 will be placed in subscript 3 since it is the “fourth” location in the array. This odd way of numbering is often confusing to new programmers; however, it quickly becomes routine.1

1 Some students actually add one more location and then ignore location 0, letting 1 be the first location. In the above example such a process would use the following definition: int agefrequency[101]; and use only the subscripts 1 through 100. Our examples will use location 0. Your instructor will tell you which method to use.

## Array Initialization

In our example, ageFrequency[0] keeps a count of how many 1s we read in, ageFrequency[1] keeps count of how many 2s we read in, etc. Thus, keeping track of how many people of a particular age exist in the data read in requires reading each age and then adding one to the location holding the count for that age. Of course it is important that all the counters start at 0. The following shows the initialization of all the elements of our sample array to 0.

for (int pos = 0; pos < TOTALYEARS; pos++)

// pos acts as the array subscript

{

ageFrequency[pos] = 0;

}

## Array Processing

A simple for loop will process the entire array, adding one to the subscript each time through the loop. Notice that the subscript (pos) starts with 0. Why is the con- dition pos < TOTALYEARS used instead of pos <= TOTALYEARS? Remember that the last subscript is one less than the total number of elements in the array. Hence the subscripts of this array go from 0 to 99.

Arrays are generally processed inside loops so that the input/output processing of each element of the array can be performed with minimal statements. Our age frequency program first needs to read in the ages from a file or from the key- board. For each age read in, the “appropriate” element of the array (the one cor- responding to that age) needs to be incremented by one. The following examples show how this can be accomplished:

**from a file using** infile **as a logical name from a keyboard with –99 as sentinel data**

cout << "Please input an age from one"

<< "to 100. input -99 to stop"

<< endl;

infile >> currentAge; cin >> currentAge; while (infile)

while (currentAge != -99)

{ {

ageFrequency[currentAge-1] = ageFrequency[currentAge-1] = ageFrequency[currentAge-1] + 1; ageFrequency[currentAge-1] + 1; infile >> currentAge; cout << "Please input an age from "

<< "one to 100. input -99 to stop"

<< endl; cin >> currentAge;

} }

The while(infile) statement means that while there is more data in the file

infile, the loop will continue to process.

To read from a file or from the keyboard we **prime the read**,2 which means the first value is read in before the test condition is checked to see if the loop

2 Priming the read for a while loop means having an input just before the loop condition (just before the while) and having another one as the last statement in the loop.

should be executed. When we read an age, we increment the location in the array that keeps track of the amount of people in the corresponding age group. Since C++ array indices always start with 0, that location will be at the subscript one value less than the age we read in.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 0 | 14 | 5 | 0 | 6 |  | 1 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | . . . . . . | 98 | 99 |
| 1 year | 2 years | 3 years | 4 years | 5 years | 6 years |  | 99 years | 100 years |

Each element of the array contains the number of people of a given age. The data shown here is from a random sample run. In writing the information stored in the array, we want to make sure that only those array elements that have values greater than 0 are output. The following code will do this.

for (int ageCounter = 0; ageCounter < TOTALYEARS; ageCounter++)

## Arrays as Arguments

if (ageFrequency[ageCounter] > 0)

cout << "The number of people " << ageCounter + 1 <<" years old is "

<< ageFrequency[ageCounter] << endl;

The for loop goes from 0 to one less than TOTALYEARS (0 to 99). This will test every element of the array. If a given element has a value greater than 0, it will be output. What does outputting ageCounter + 1 do? It gives the age we are deal- ing with at any given time, while the value of ageFrequency[ageCounter] gives the number of people in that age group.

The complete age frequency program will be given as one of the lab assign- ments in Lab 7.4.

Arrays can be passed as arguments (parameters) to functions. Although variables can be passed by value or reference, arrays are always **passed by pointer**, which is similar to pass by reference, since it is not efficient to make a “copy” of all ele- ments of the array. Pass by pointer is discussed further in Lesson Set 9. This means that arrays, like pass by reference parameters, can be altered by the call- ing function. However, they NEVER have the & symbol between the data type and name, as pass by reference parameters do. Sample Program 7.1 illustrates how arrays are passed as arguments to functions.

*Sample Program 7.1:*

// The grade average program

// This program illustrates how one-dimensional arrays are used and how

// they are passed as arguments to functions. It contains two functions.

// The first function is called to allow the user to input a set of grades and

// store them in an array. The second function is called to find the average

// grade.

#include <iostream> using namespace std;

const int TOTALGRADES = 50; // TOTALGRADES is the maximum size of the array

// function prototypes

void getData(int array[], int& sizeOfArray);

// the procedure that will read values into the array

float findAverage(const int array[], int sizeOfArray);

// the procedure that will find the average of values

// stored in an array. The word const in front of the

// data type of the array prevents the function from

// altering the array

int main()

|  |  |  |
| --- | --- | --- |
| { |  |  |
| int grades[TOTALGRADES]; | // | defines an array that holds up to 50 ints |
| int numberOfGrades = 0; | // | the number of grades read in |
| float average; | // | the average of all grades read in |

getData(grades, numberOfGrades); // getData is called to read the grades into

// the array and store how many grades there

// are in numberOfGrades average = findAverage(grades, numberOfGrades);

cout << endl << "The average of the " << numberOfGrades

<< " grades read in is " << average << "." << endl << endl;

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// getData

//

// task: This function inputs and stores data in the grades array.

// data in: none (the parameters contain no information needed by the

// getData function)

// data out: an array containing grades and the number of grades

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void getData(int array[], int& sizeOfArray)

{

int pos = 0; // array index which starts at 0

int grade; // holds each individual grade read in

cout << "Please input a grade or type -99 to stop: " << endl; cin >> grade;

### continues

while (grade != -99)

{

array[pos] = grade; // store grade read in to next array location pos ++; // increment array index

cout << "Please input a grade or type -99 to stop: " << endl; cin >> grade;

}

sizeOfArray = pos; // upon exiting the loop, pos holds the

// number of grades read in, which is sent

// back to the calling function

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// findAverage

//

// task: This function finds and returns the average of the values

//

// data in: the array containing grades and the array size

// data returned: the average of the grades contained in that array

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

float findAverage (const int array[], int sizeOfArray)

{

int sum = 0; // holds the sum of all grades in the array

for (int pos = 0; pos < sizeOfArray; pos++)

{

sum = sum + array[pos];

// add grade in array position pos to sum

}

return float(sum)/sizeOfArray;

}

Notice that a set of empty brackets [ ] follows the parameter of an array which indicates that the data type of this parameter is in fact an array. Notice also that no brackets appear in the call to the functions that receive the array.

Since arrays in C++ are passed by pointer, which is similar to pass by reference, it allows the original array to be altered, even though no & is used to designate this. The getData function is thus able to store new values into the array. There may be times when we do not want the function to alter the values of the array. Inserting the word **const** before the data type on the formal parameter list pre- vents the function from altering the array even though it is passed by pointer. This is why in the preceding sample program the findAverage function and header had the word const in front of the data type of the array.

float findAverage (const int array[], int sizeOfArray); // prototype float findAverage (const int array[], int sizeOfArray) // function header

The variable numberOfGrades contains the number of elements in the array to be processed. In most cases not every element of the array is used, which means the size of the array given in its definition and the number of actual elements used are rarely the same. For that reason we often pass the actual number of ele- ments used in the array as a parameter to a procedure that uses the array. The variable numberOfGrades is explicitly passed by reference (by using &) to the getData function where its corresponding formal parameter is called sizeOfArray.

Prototypes can be written without named parameters. Function headers must include named parameters.

float findAverage (const int [], int); // prototype without named parameters

The use of brackets in function prototypes and headings can be avoided by declaring a programmer defined data type. This is done in the global section with a **typedef** statement.

*Example:* typedef int GradeType[50];

This declares a data type, called GradeType, that is an array containing 50 inte- ger memory locations. Since GradeType is a data type, it can be used in defining variables. The following defines grades as an integer array with 50 elements.

GradeType grades;

It has become a standard practice (although not a requirement) to use an upper- case letter to begin the name of a data type. It is also helpful to include the word “type” in the name to indicate that it is a data type and not a variable.

Sample Program 7.2 shows the revised code (in bold) of Sample Program 7.1 using

typedef.

*Sample Program 7.2:*

// Grade average program

// This program illustrates how one-dimensional arrays are used and how

// they are passed as arguments to functions. It contains two functions.

// The first function is called to input a set of grades and store them

// in an array. The second function is called to find the average grade.

#include <iostream> using namespace std;

const int TOTALGRADES = 50; // maximum size of the array

// function prototypes

**typedef int GradeType[TOTALGRADES];** // declaration of an integer array data type

// called GradeType

### continues

void getData(**GradeType array**, int& sizeOfArray);

// the procedure that will read values into the array

float findAverage(const **GradeType array**, int sizeOfArray);

// the procedure that will find the average of values

// stored in an array. The word const in front of the

// data type of the array prevents the function from

// altering the array

int main()

{

**GradeType grades;** // defines an array that holds up to 50 ints int numberOfGrades = 0; // the number of grades read in

float average; // the average of all grades read in

getData(grades, numberOfGrades);// getData is called to read the grades into

// the array and store how many grades there

// are in numberOfGrades average = findAverage(grades, numberOfGrades);

cout << endl << "The average of the " << numberOfGrade

<< " grades read in is " << average << "." << endl << endl;

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// getData

//

// task: This function inputs and stores data in the grades array.

// data in: none

// data out: an array containing grades and the number of grades

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void getData(**GradeType array**, int& sizeOfArray)

{

int pos = 0; // array index which starts at 0

int grade; // holds each individual grade read in

cout << "Please input a grade or type -99 to stop: " << endl; cin >> grade;

while (grade != -99)

{

array[pos] = grade; // store grade read in to next array location pos ++; // increment array index

cout << "Please input a grade or type -99 to stop: " << endl; cin >> grade;

}

sizeOfArray = pos; // upon exiting the loop, pos holds the

// number of grades read in, which is sent

// back to the calling function

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// findAverage

//

// task: This function finds and returns the average of the values

//

// data in: the array containing grades and the array size

// data returned: the average of the grades contained in that array

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

float findAverage (const **GradeType array**, int sizeOfArray)

{

int sum = 0; // holds the sum of all grades in the array

for (int pos = 0; pos < sizeOfArray; pos++)

{

sum = sum + array[pos];

// add grade in array position pos to sum

}

return float(sum)/sizeOfArray;

}

## Two-Dimensional Arrays

This method of using typedef to eliminate brackets in function prototypes and headings is especially useful for multi-dimensional arrays such as those intro- duced in the next section.

Data is often contained in a table of rows and columns that can be implement- ed with a two-dimensional array. Suppose we want to read data representing profits (in thousands) for a particular year and quarter.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Quarter | 1 | Quarter | 2 | Quarter | 3 | Quarter 4 |
| 72 |  | 80 |  | 10 |  | 100 |
| 82 |  | 90 |  | 43 |  | 42 |
| 10 |  | 87 |  | 48 |  | 53 |

This can be done using a **two-dimensional array**.

*Example:*

const NO\_OF\_ROWS = 3; const NO\_OF\_COLS = 4;

typedef float ProfitType[NO\_OF\_ROWS][NO\_OF\_COLS]; //declares a new data type

//which is a 2 dimensional

//array of floats

### continues

int main()

{

ProfitType profit; // defines profit as a 2 dimensional array

for (int row\_pos = 0; row\_pos < NO\_OF\_ROWS; row\_pos++) for (int col\_pos = 0; col\_pos < NO\_OF\_COLS; col\_pos++)

{

cout << "Please input a profit" << endl; cin >> profit[row\_pos][col\_pos];

}

return 0;

}

## Multi-Dimensional Arrays

Arrays of Strings

A two dimensional array normally uses two loops (one nested inside the other) to read, process, or output data.

How many times will the code above ask for a profit? It processes the inner loop NO\_OF\_ROWS \* NO\_OF\_COLS times, which is 12 times in this case.

C++ arrays can have any number of dimensions (although more than three is rarely used). To input, process or output every item in an *n*-dimensional array, you need *n* nested loops.

Any variable defined as **char** holds only one character. To hold more than one character in a single variable, that variable needs to be an array of characters. A string (a group of characters that usually form meaningful names or words) is really just an array of characters. A complete lesson on characters and strings is given in Lesson Set 10.

# P R E - LA B W R I T I N G A S S I G N M E N T

## Fill-in-the-Blank Questions

1. The first subscript of every array in C++ is and the last is

less than the total number of locations in the array.

1. The amount of memory allocated to an array is based on the

and the of locations or size of the array.

1. Array initialization and processing is usually done inside a

.

1. The statement can be used to declare an array type and is often used for multidimensional array declarations so that when passing arrays as parameters, brackets do not have to be used.
2. Multi-dimensional arrays are usually processed within loops.
3. Arrays used as arguments are always passed by .

# L E S S O N 7 A

1. In passing an array as a parameter to a function that processes it, it is often necessary to pass a parameter that holds the of

used in the array.

1. A string is an array of .
2. Upon exiting a loop that reads values into an array, the variable used as a(n) to the array will contain the size of that array.
3. An *n*-dimensional array will be processed within nested loops when accessing all members of the array.

## LAB 7.1 Working with One-Dimensional Arrays

Retrieve program testscore.cpp from the Lab 7 folder. The code is as follows:

// This program will read in a group of test scores (positive integers from 1 to 100)

// from the keyboard and then calculate and output the average score

// as well as the highest and lowest score. There will be a maximum of 100 scores.

#### // PLACE YOUR NAME HERE

#include <iostream> using namespace std;

typedef int GradeType[100]; // declares a new data type:

// an integer array of 100 elements

float findAverage (const GradeType, int); // finds average of all grades int findHighest (const GradeType, int); // finds highest of all grades int findLowest (const GradeType, int); // finds lowest of all grades

int main()

{

GradeType grades; // the array holding the grades.

int numberOfGrades; // the number of grades read.

int pos; // index to the array.

float avgOfGrades; // contains the average of the grades.

int highestGrade; // contains the highest grade.

int lowestGrade; // contains the lowest grade.

// Read in the values into the array pos = 0;

cout << "Please input a grade from 1 to 100, (or -99 to stop)" << endl;

### continues

cin >> grades[pos];

while (grades[pos] != -99)

{

#### // Fill in the code to read the grades

}

numberOfGrades = ; **// Fill blank with appropriate identifier**

// call to the function to find average avgOfGrades = findAverage(grades, numberOfGrades);

cout << endl << "The average of all the grades is " << avgOfGrades << endl;

#### // Fill in the call to the function that calculates highest grade

cout << endl << "The highest grade is " << highestGrade << endl;

#### // Fill in the call to the function that calculates lowest grade

**// Fill in code to write the lowest to the screen**

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// findAverage

//

// task: This function receives an array of integers and its size.

// It finds and returns the average of the numbers in the array

// data in: array of floating point numbers

// data returned: average of the numbers in the array

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* float findAverage (const GradeType array, int size)

{

float sum = 0; // holds the sum of all the numbers for (int pos = 0; pos < size; pos++)

sum = sum + array[pos];

return (sum / size); //returns the average

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// findHighest

//

// task: This function receives an array of integers and its size.

// It finds and returns the highest value of the numbers in the array

// data in: array of floating point numbers

// data returned: highest value of the numbers in the array

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* int findHighest (const GradeType array, int size)

{

#### / Fill in the code for this function

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// findLowest

//

// task: This function receives an array of integers and its size.

// It finds and returns the lowest value of the numbers in the array

// data in: array of floating point numbers

// data returned: lowest value of the numbers in the array

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* int findLowest (const GradeType array, int size)

{

#### // Fill in the code for this function

}

*Exercise 1:* Complete this program as directed.

*Exercise 2:* Run the program with the following data: 90 45 73 62 -99 and record the output here:

*Exercise 3:* Modify your program from Exercise 1 so that it reads the informa- tion from the gradfile.txt file, reading until the end of file is encoun- tered. You will need to first retrieve this file from the Lab 7 folder and place it in the same folder as your C++ source code. Run the program.

## Lab 7.2 Strings as Arrays of Characters

Retrieve program student.cpp from the Lab 7 folder.

// This program will input an undetermined number of student names

// and a number of grades for each student. The number of grades is

// given by the user. The grades are stored in an array.

// Two functions are called for each student.

// One function will give the numeric average of their grades.

// The other function will give a letter grade to that average.

// Grades are assigned on a 10 point spread.

// 90-100 A 80-89 B 70-79 C 60-69 D Below 60 F

#### // PLACE YOUR NAME HERE

#include <iostream> #include <iomanip> using namespace std;

const int MAXGRADE = 25; // maximum number of grades per student const int MAXCHAR = 30; // maximum characters used in a name

typedef char StringType30[MAXCHAR + 1];// character array data type for names

// having 30 characters or less.

typedef float GradeType[MAXGRADE]; // one dimensional integer array data type

float findGradeAvg(GradeType, int); // finds grade average by taking array of

// grades and number of grades as parameters

char findLetterGrade(float); // finds letter grade from average given

// to it as a parameter

int main()

{

|  |  |  |
| --- | --- | --- |
| StringType30 firstname, | lastname;// | two arrays of characters defined |
| int numOfGrades; | // | holds the number of grades |
| GradeType grades; | // | grades defined as a one dimensional array |
| float average; | // | holds the average of a student's grade |
| char moreInput; | // | determines if there is more input |

cout << setprecision(2) << fixed << showpoint;

// Input the number of grades for each student

cout << "Please input the number of grades each student will receive." << endl

<< "This must be a number between 1 and " << MAXGRADE << " inclusive”

<< endl;

cin >> numOfGrades;

while (numOfGrades > MAXGRADE || numOfGrades < 1)

{

cout << "Please input the number of grades for each student." << endl

<< "This must be a number between 1 and " << MAXGRADE

<< " inclusive\n"; cin >> numOfGrades;

}

// Input names and grades for each student

cout << "Please input a y if you want to input more students"

<< " any other character will stop the input" << endl; cin >> moreInput;

while (moreInput == 'y' || moreInput == 'Y')

{

cout << "Please input the first name of the student" << endl; cin >> firstname;

cout << endl << "Please input the last name of the student" << endl; cin >> lastname;

for (int count = 0; count < numOfGrades; count++)

{

cout << endl << "Please input a grade" << endl;

#### // Fill in the input statement to place grade in the array

}

cout << firstname << " " << lastname << " has an average of ";

#### // Fill in code to get and print average of student to screen

**// Fill in call to get and print letter grade of student to screen**

cout << endl << endl << endl;

cout << "Please input a y if you want to input more students"

<< " any other character will stop the input" << endl; cin >> moreInput;

}

return 0;

}

### continues

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

|  |  |  |
| --- | --- | --- |
| //  // |  | findGradeAvg |
| // | task: | This function finds the average of the |
| // |  | numbers stored in an array. |
| // |  |  |
| // | data in: | an array of integer numbers |
| // | data returned: | the average of all numbers in the array |
| // |  |  |

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* float findGradeAvg(GradeType array, int numGrades)

{

#### // Fill in the code for this function

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// findLetterGrade

//

// task: This function finds the letter grade for the number

// passed to it by the calling function

//

// data in: a floating point number

// data returned: the grade (based on a 10 point spread) based on the number

// passed to the function

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* char findLetterGrade(float numGrade)

{

#### // Fill in the code for this function

}

*Exercise 1:* Complete the program by filling in the code. (Areas in bold)

Run the program with 3 grades per student using the sample data below.

Mary Brown 100 90 90

|  |  |
| --- | --- |
| George Smith | 90 30 50 |
| Dale Barnes | 80 78 82 |
| Sally Dolittle | 70 65 80 |
| Conrad Bailer | 60 58 71 |

You should get the following results:

**Mary Brown has an average of 93.33 which gives the letter grade of A George Smith has an average of 56.67 which gives the letter grade of F Dale Barnes has an average of 80.00 which gives the letter grade of B Sally Dolittle has an average of 71.67 which gives the letter grade of C Conrad Bailer has an average of 63.00 which gives the letter grade of D**

# L E S S O N 7 B

## LAB 7.3 Working with Two-Dimensional Arrays

Look at the following table containing prices of certain items:

|  |  |  |  |
| --- | --- | --- | --- |
| 12.78 | 23.78 | 45.67 | 12.67 |
| 7.83 | 4.89 | 5.99 | 56.84 |
| 13.67 | 34.84 | 16.71 | 50.89 |

These numbers can be read into a two-dimensional array.

Retrieve price.cpp from the Lab 7 folder. The code is as follows:

// This program will read in prices and store them into a two-dimensional array.

// It will print those prices in a table form.

#### // PLACE YOUR NAME HERE

#include <iostream> #include <iomanip> using namespace std;

const MAXROWS = 10; const MAXCOLS = 10;

typedef float PriceType[MAXROWS][MAXCOLS]; // creates a new data type

// of a 2D array of floats

|  |  |  |
| --- | --- | --- |
| void | getPrices(PriceType, int&, int&); | // gets the prices into the array |
| void | printPrices(PriceType, int, int); | // prints data as a table |

int main()

{

int rowsUsed; // holds the number of rows used

int colsUsed; // holds the number of columns used

PriceType priceTable; // a 2D array holding the prices

getPrices(priceTable, rowsUsed, colsUsed); // calls getPrices to fill the array printPrices(priceTable, rowsUsed, colsUsed);// calls printPrices to display array

return 0;

}

### continues

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// getPrices

//

// task: This procedure asks the user to input the number of rows and

// columns. It then asks the user to input (rows \* columns) number of

// prices. The data is placed in the array.

// data in: none

// data out: an array filled with numbers and the number of rows

// and columns used.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void getPrices(PriceType table, int& numOfRows, int& numOfCols)

{

cout << "Please input the number of rows from 1 to "<< MAXROWS << endl; cin >> numOfRows;

cout << "Please input the number of columns from 1 to "<< MAXCOLS << endl; cin >> numOfCols;

for (int row = 0; row < numOfRows; row++)

{

for (int col = 0; col < numOfCols; col++)

#### // Fill in the code to read and store the next value in the array

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// printPrices

//

// task: This procedure prints the table of prices

// data in: an array of floating point numbers and the number of rows

// and columns used.

// data out: none

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void printPrices(PriceType table, int numOfRows, int numOfCols)

{

cout << fixed << showpoint << setprecision(2); for (int row = 0; row < numOfRows; row++)

{

for (int col = 0; col < numOfCols; col++)

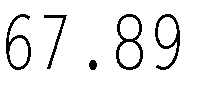
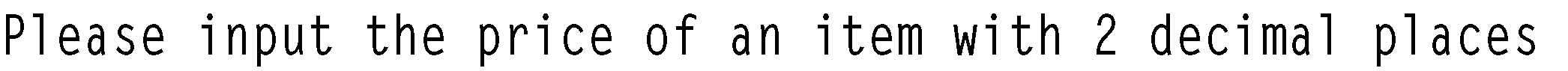
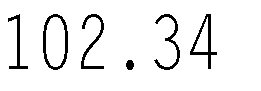
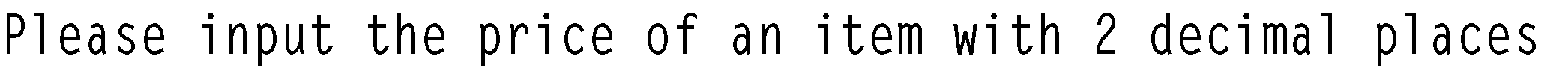
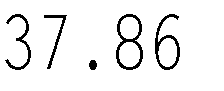
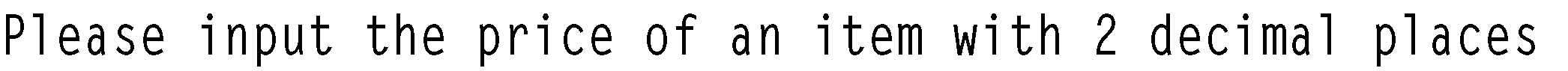
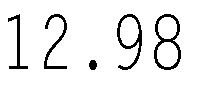
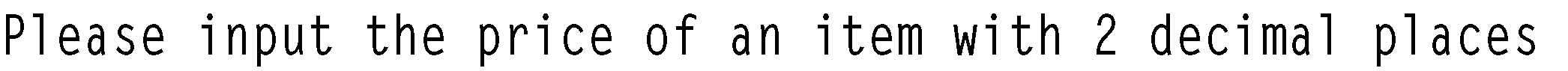
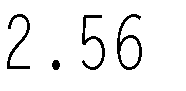
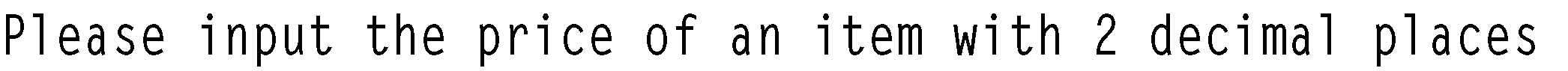
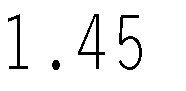
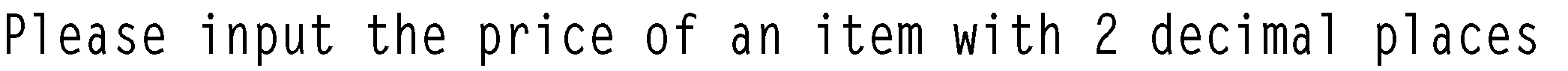
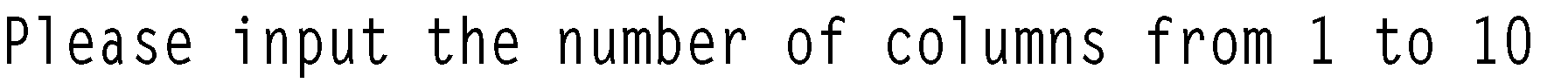
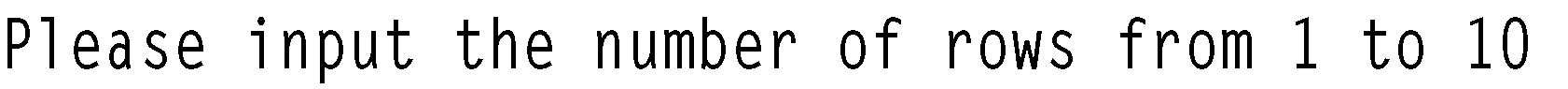
#### // Fill in the code to print the table

}

}

*Exercise 1:* Fill in the code to complete both functions getPrices and

printPrices, then run the program with the following data:



|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

*Exercise 2:* Why does getPrices have the parameters numOfRows and numOfCols passed by reference whereas printPrices has those parameters passed by value?

*Exercise 3:* The following code is a function that returns the highest price in the array. After studying it very carefully, place the function in the above program and have the program print out the highest value.

float findHighestPrice(PriceType table, int numOfRows, int numOfCols)

// This function returns the highest price in the array

{

float highestPrice;

highestPrice = table[0][0]; // make first element the highest price for (int row = 0; row < numOfRows; row++)

for (int col = 0; col < numOfCols; col++) if ( highestPrice < table[row][col] )

highestPrice = table[row][col];

return highestPrice;

}

*continues*

NOTE: This is a value returning function. Be sure to include its prototype in the global section.

*Exercise 4:* Create another value returning function that finds the lowest price in the array and have the program print that value.

*Exercise 5:* After completing all the exercises above, run the program again with the values from Exercise 1 and record your results.

*Exercise 6:* (Optional) Look at the following table that contains quarterly sales transactions for three years of a small company. Each of the quarterly transactions are integers (number of sales) and the year is also an integer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| YEAR | Quarter 1 | Quarter 2 | Quarter 3 | Quarter 4 |
| 2000 | 72 | 80 | 60 | 100 |
| 2001 | 82 | 90 | 43 | 98 |
| 2002 | 64 | 78 | 58 | 84 |

We could use a two-dimensional array consisting of 3 rows and 5 columns. Even though there are only four quarters we need 5 columns (the first column holds the year).

Retrieve quartsal.cpp from the Lab 7 folder. The code is as follows:

// This program will read in the quarterly sales transactions for a given number

// of years. It will print the year and transactions in a table format.

// It will calculate year and quarter total transactions.

#### // PLACE YOUR NAME HERE

#include <iostream> #include <iomanip> using namespace std;

const MAXYEAR = 10; const MAXCOL = 5;

typedef int SalesType[MAXYEAR][MAXCOL]; // creates a new 2D integer data type

void getSales(SalesType, int&); // places sales figures into the array

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| void | printSales(SalesType, | int); | // | prints data as a table |
| void | printTableHeading(); |  | // | prints table heading |

int main()

{

int yearsUsed; // holds the number of years used

SalesType sales; // 2D array holding

// the sales transactions

getSales(sales, yearsUsed); // calls getSales to put data in array

printTableHeading(); // calls procedure to print the heading printSales(sales, yearsUsed); // calls printSales to display table

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

|  |  |  |
| --- | --- | --- |
| // |  | printTableHeading |
| // | task: | This procedure prints the table heading |
| // | data in: | none |
| // | data out: | none |
| // |  |  |

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void printTableHeading()

{

cout << setw(30) << "YEARLY QUARTERLY SALES" << endl << endl << endl;

cout << setw(10) << "YEAR" << setw(10) << "Quarter 1"

<< setw(10) << "Quarter 2" << setw(10) << "Quarter 3"

<< setw(10) << "Quarter 4" << endl;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// getSales

//

// task: This procedure asks the user to input the number of years.

// For each of those years it asks the user to input the year

// (e.g. 2004), followed by the sales figures for each of the

// 4 quarters of that year. That data is placed in a 2D array

// data in: a 2D array of integers

// data out: the total number of years

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void getSales(SalesType table, int& numOfYears)

{

cout << "Please input the number of years (1-" << MAXYEAR << ')' << endl; cin >> numOfYears;

**// Fill in the code to read and store the next value**

### continues

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

|  |  |  |
| --- | --- | --- |
| //  // |  | printSales |
| // | task: | This procedure prints out the information in the array |
| // | data in: | an array containing sales information |
| // | data out: | none |
| // |  |  |

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void printSales(SalesType table, int numOfYears)

{

#### // Fill in the code to print the table

}

Fill in the code for both getSales and printSales.

This is similar to the price.cpp program in Exercise 1; however, the code will be different. This is a table that contains something other than sales in column one.

*Exercise 7:* Run the program so that the chart from Exercise 6 is printed.

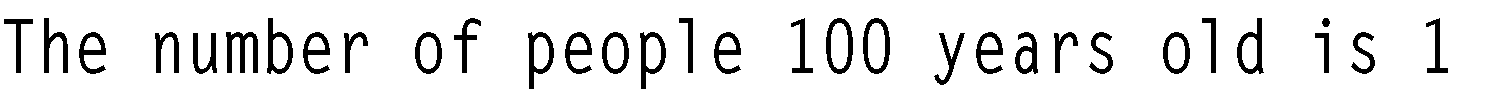
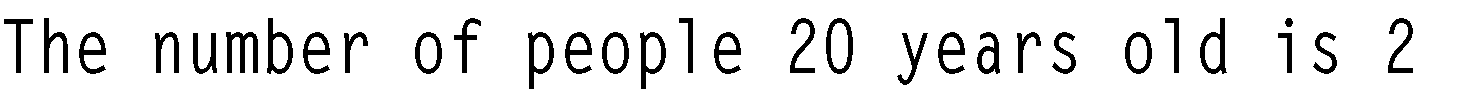
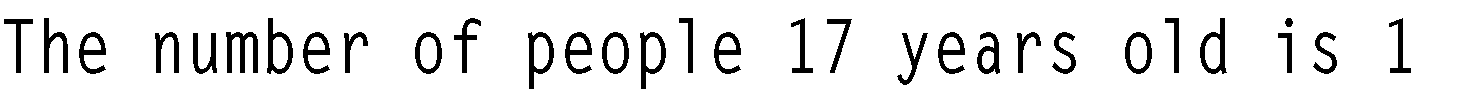
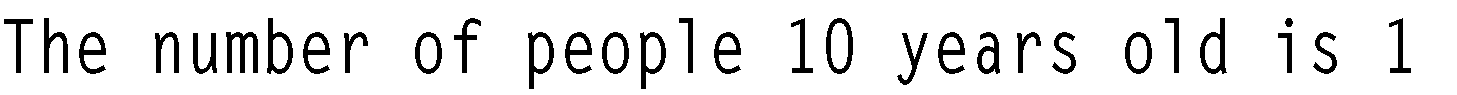
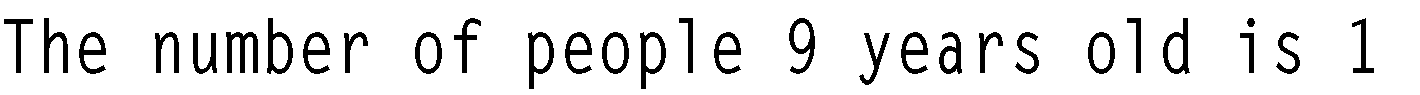
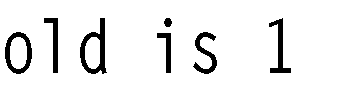
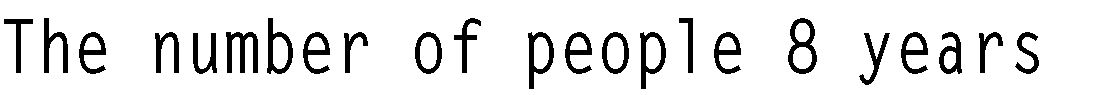
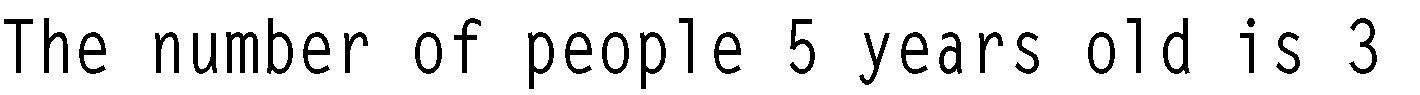
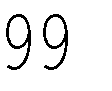
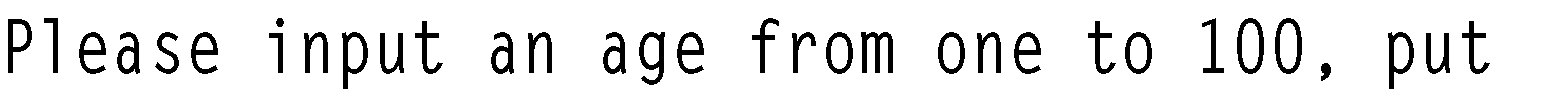
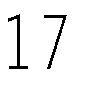
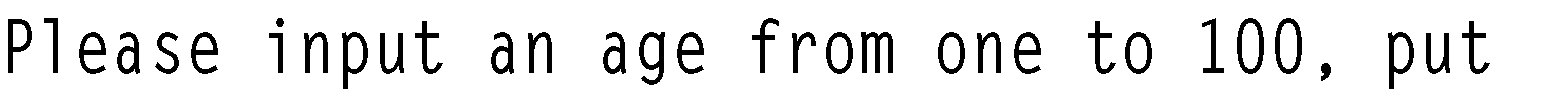
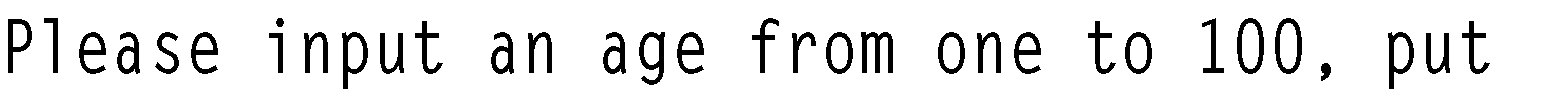
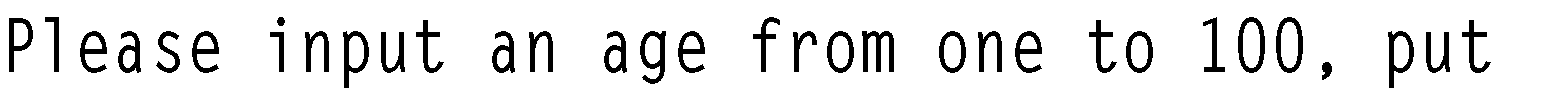
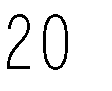
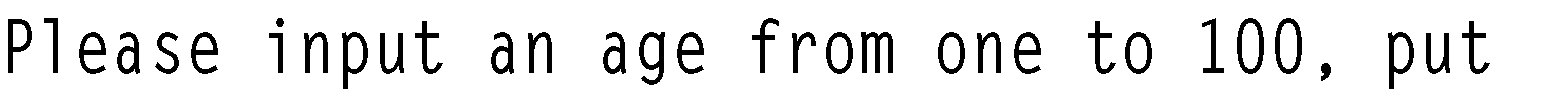
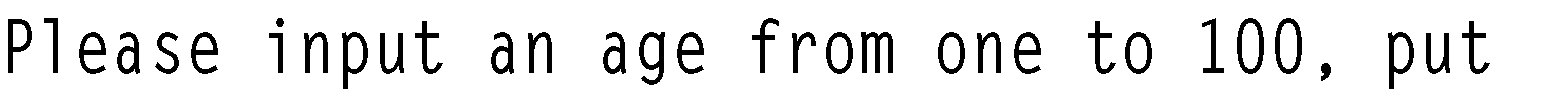
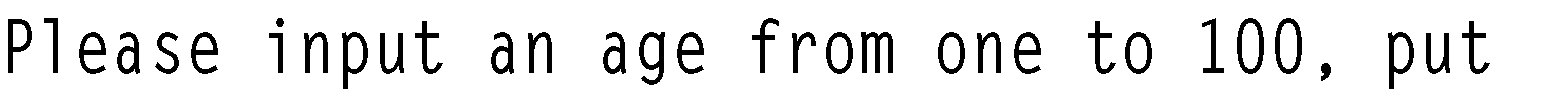
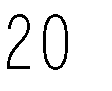
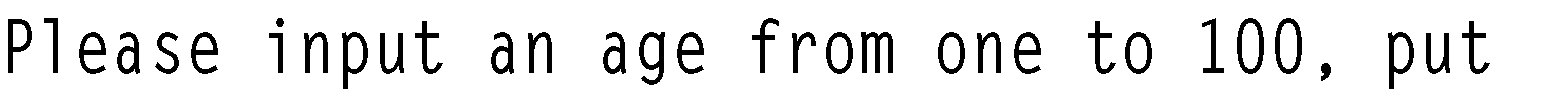
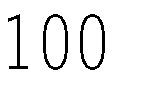
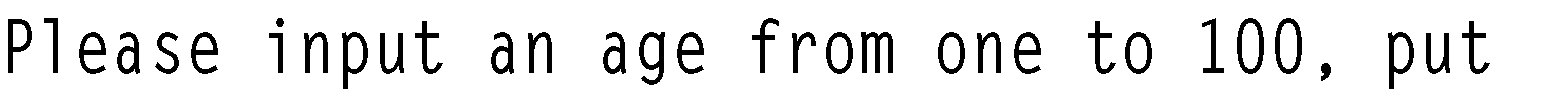
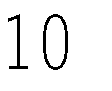
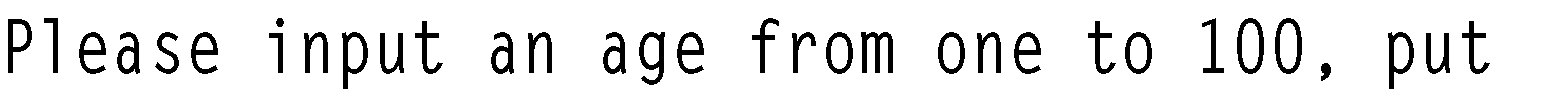
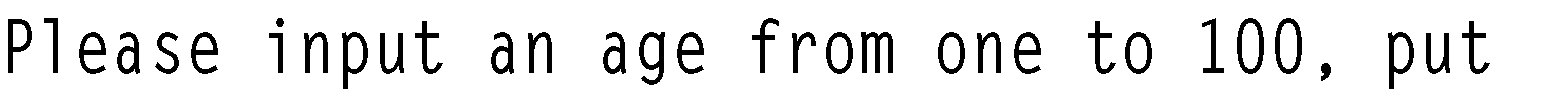
## LAB 7.4 Student Generated Code Assignments

*Option 1:* Write the complete age population program given in the Pre-lab Reading Assignment.

Statement of the problem:

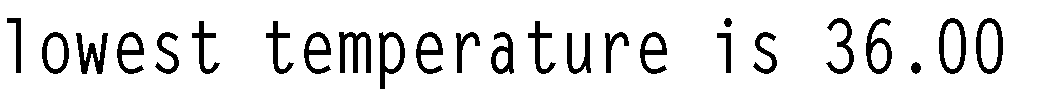
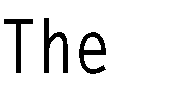
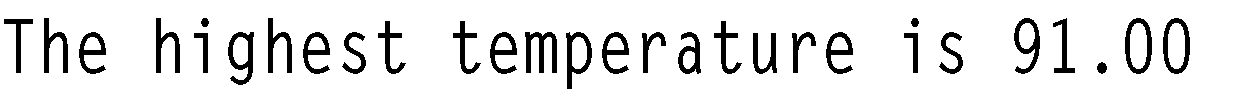
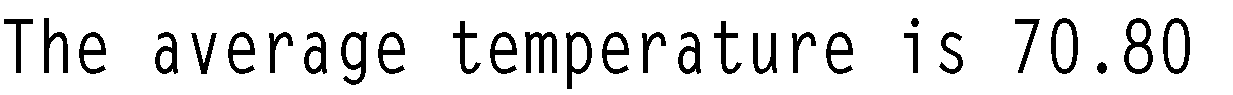
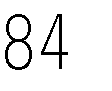
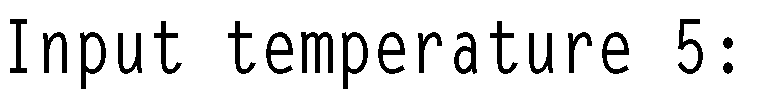
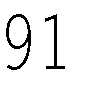
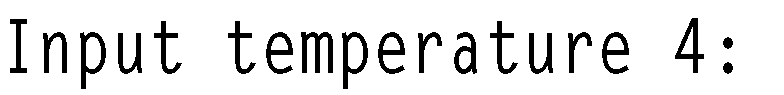
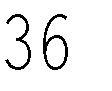
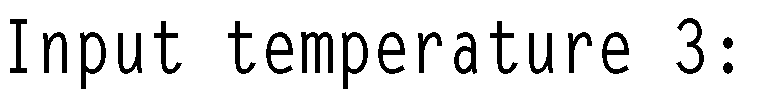
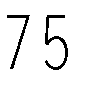
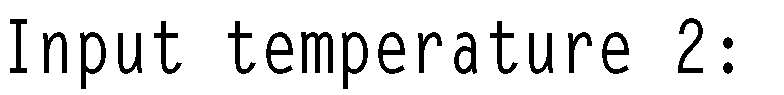
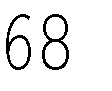
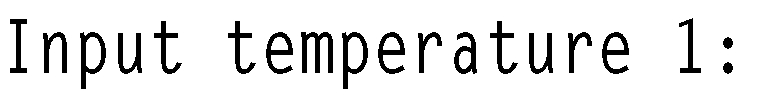
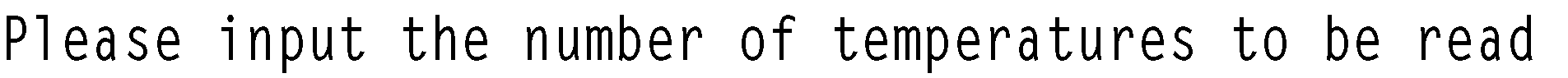
Given a list of ages (1 to 100) from the keyboard, the program will tally how many people are in each age group.

*Sample Run:*

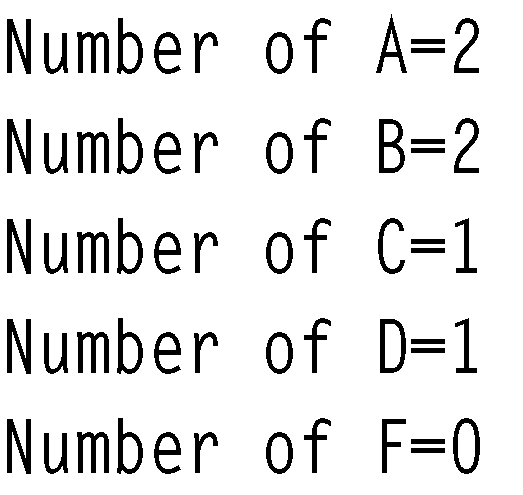


*Option 2:* Write a program that will input temperatures for consecutive days. The program will store these values into an array and call a function that will return the average of the temperatures. It will also call a function that will return the highest temperature and a function that will return the lowest temperature. The user will input the number of temperatures to be read. There will be no more than 50 temperatures. Use typedef to declare the array type. The average should be displayed to two decimal places.

*Sample Run:*



*Option 3:* Write a program that will input letter grades (A, B, C, D, F), the number of which is input by the user (a maximum of 50 grades). The grades will be read into an array. A function will be called five times (once for each letter grade) and will return the total number of grades in that category. The input to the function will include the array, number of elements in the array and the letter category (A, B, C, D or F). The pro- gram will print the number of grades that are A, B, etc.

*Sample Run:*

