

CMPS 150 Workbook: Chapter 5

Section 5.1: Introduction to Arrays

1) What is the output of the following code?

```
const int MAX = 5;
double a[MAX];

a[0] = 67.0;
a[1] = 23.1;
a[2] = -17.4;
a[3] = 55.2;
a[4] = -5.9;

for (int x=0; x<MAX; x++)
    cout << a[x] << ' ';
cout << endl;
```

2) What is the output of the following code?

```
const int LIMIT = 6;
int r[LIMIT];

r[0] = 34;
r[1] = -6;
r[2] = 17;
r[3] = 24;
r[4] = -41;
r[5] = 99;

for (int a=LIMIT-1; a>=0; a--)
    cout << r[a] << ' ';
cout << endl;
```

3) What is the output of the following code?

```
const int SIZE = 100;
int squares[SIZE];

for (int m=0; m<SIZE; m++)
    squares[m] = m * m;
for (int m=0; m<6; m += 2)
    cout << squares[m] << ' ';
cout << endl;
```

4) What is the output of the following code?

```
const int SIZE = 100;
int squares[SIZE];

for (int m=0; m<SIZE; m++)
    squares[m] = m * m;
for (int m=0; m<10; m += 4)
    cout << squares[m];
cout << endl;
```

5) What is the output of the following code?

```
bool truth[] = {true, true, false, true};
int t=0, f=0;

for (int x=0; x<4; x++)
    if (truth[x])
        t++;
    else
        f++;
cout << t << ' ' << f << endl;
```

6) What is the output of the following code?

```
string names[] = {"Sue", "Sam", "Fred", "Pam", "Joy"};

for (int a=4; a>=0; a--)
    cout << names[a] << endl;
```

7) Declare arrays for the following:

- a) 100 temperatures
- b) 53 weights
- c) 27 street addresses

8) Declare arrays for the following:

- a) attendance counters for 10 movie houses
- b) 97 arrival times for airline flights
- c) 421 company names

9) Using a loop, write code to output all the values stored in this array. (Assume that every element in the array contains a value.)

```
const int MAX_FISHING = 1000;
int catches[MAX_FISHING];
```

10) Using a loop, write code to output all the values stored in this array. (Assume that every element in the array contains a value.)

```
const int MAX_BANANAS = 50000;
double loads[MAX_BANANAS];
```

11) Write the code to find the smallest (minimum) value stored in this array. (Assume that every element in the array contains a value.)

```
const int MAX_FISHING = 1000;
int catches[MAX_FISHING];
```

12) Write the code to find the largest (maximum) value stored in this array. (Assume that every element in the array contains a value.)

```
const int MAX_BANANAS = 50000;
double loads[MAX_BANANAS];
```

Section 5.1 Additional: Arrays of Structs

1) Given the struct

```
struct Dinosaur {
    string name;
    double weight;
    double length;
    bool carnivorous;
};
```

what is the output of the following code?

```
const int MAX_DINO = 10;
Dinosaur d[MAX_DINO];
d[0].name = "Barney";
d[0].weight = 12.5;
d[1].name = "Dino";
d[1].weight = .15;
cout << d[1].name << ' ' << d[0].weight << endl;
cout << d[0].name << ' ' << d[1].weight << endl;
```

2) Given the struct

```
struct Phone
{
    string firstName;
    string lastName;
    string phoneNumber;
};
```

what is the output of the following code?

```
const int MAX_PHONES = 100;
Phone phoneBook[MAX_PHONES];

phoneBook[0].firstName = "Max";
phoneBook[0].lastName = "Headroom";
phoneBook[0].phoneNumber = "999-999-9999";
phoneBook[4].firstName = "Black";
phoneBook[4].lastName = "Adder";
phoneBook[4].phoneNumber = "1039-1918";

cout << phoneBook[0].firstName << ' '
      << phoneBook[4].lastName << ' '
      << phoneBook[0].phoneNumber << endl;
```

3) Given the struct

```
struct Soil
{
    double sampleWeight;
    double clayWeight;
    double sandWeight;
    double waterWeight;
};
```

declare an array of type *Soil* that can contain 400 soil samples.

4) Given the struct

```
struct Catch
{
    double catchWeight;
    int numberFish;
};
```

declare an array of type *Catch* that can contain information on 1000 catches.

5) Using the array you declared in #3 of this section and assuming that every element in the array contains data, write the code necessary to output all data in the array.

6) Using the array you declared in #4 of this section and assuming that every element in the array contains data, write the code necessary to output all data in the array.

Section 5.1 Additional: Arrays of Classes

1) Given the following class definition

```
class Pixel
{
public:
    Pixel();
    Pixel(int initRed, int initGreen, int initBlue);
    int ReturnRed();
    int ReturnGreen();
    int ReturnBlue();
private:
    int red, green, blue;
};

Pixel::Pixel()
{
    red    = 0;
    green  = 0;
    blue   = 0;
}

Pixel::Pixel(int initRed, int initGreen, int initBlue)
{
    red    = initRed;
    green  = initGreen;
    blue   = initBlue;
}

int Pixel::ReturnRed()
{
    return red;
}

int Pixel::ReturnGreen()
{
    return green;
}
```

```

int Pixel::ReturnBlue()
{
    return blue;
}

```

what is the output of the following code?

```

const int MAX = 100000;
Pixel dots[MAX];
for (int x=0; x<100; x++)
    dots[x] = Pixel(x,x+1, x+2);
cout << dots[0].ReturnRed() << ' '
     << dots[1].ReturnGreen() << ' '
     << dots[2].ReturnBlue () << endl;

```

2) Given the following class

```

class Triangle
{
public:
    Triangle();
    Triangle(int iX1, int iY1, int iX2, int iY2, int iX2, int iY2);
    void ReturnPair1(int &x, int &y);
    void ReturnPair2(int &x, int &y);
    void ReturnPair3(int &x, int &y);
    bool IsSameAs(Triangle t);
private:
    int x1, y1, x2, y2, x3, y3;
};

Triangle::Triangle ()
{
    x1 = x2 = x3 = y1 = y2 = y3 = 0;
}

Triangle::Triangle(int iX1, int iY1,
                  int iX2, int iY2,
                  int iX2, int iY2)
{
    x1 = iX1;
    y1 = iY1;
    x2 = iX2;
    y2 = iY2;
    x3 = iX3;
    y3 = iY3;
}

```

```
void Triangle::ReturnPair1(int &x, int &y)
{
    x = x1;
    y = y1;
}

void Triangle::ReturnPair2(int &x, int &y)
{
    x = x2;
    y = y2;
}

void Triangle::ReturnPair3(int &x, int &y)
{
    x = x3;
    y = y3;
}

bool Triangle::IsSameAs(Triangle t)
(
    int tx1, ty1, tx2, ty2, tx3, ty3;
    t.ReturnPair1(tx1, ty1);
    t.ReturnPair2(tx2, ty2);
    t.ReturnPair3(tx3, ty3);

    if (x1==tx1 && x2==tx2 && x3==tx3 &&
        y1==ty1 && y2==ty2 && y3==ty3)
        return true;
    if (x1==tx2 && x2==tx3 && x3==tx1 &&
        y1==ty2 && y2==ty3 && y3==ty1)
        return true;
    if (x1==tx3 && x2==tx1 && x3==tx2 &&
        y1==ty3 && y2==ty1 && y3==ty2)
        return true;
    if (x1==tx3 && x2==tx2 && x3==tx1 &&
        y1==ty3 && y2==ty2 && y3==ty3)
        return true;
    if (x1==tx2 && x2==tx1 && x3==tx3 &&
        y1==ty2 && y2==ty1 && y3==ty3)
        return true;
    if (x1==tx1 && x2==tx3 && x3==tx2 &&
        y1==ty1 && y2==ty3 && y3==ty2)
        return true;

    return false;
}
```

What is the output of the following code?

```
const int MAX_TRI = 300;
Triangle t[MAX_TRI];
int p1x, p1y, p2x, p2y, p3x, p3y;

for (int x=0; x<100; x++)
    t[x] = Triangle(x,x, x+1,x+1, x+1,x);

t[0].ReturnPair1(p1x, p1y);
t[1].ReturnPair2(p2x, p2y);
t[2].ReturnPair2(p3x, p3y);

cout << '(' << p1x << ',' << p1y << ')' << endl;
cout << '(' << p2x << ',' << p2y << ')' << endl;
cout << '(' << p3x << ',' << p3y << ')' << endl;
```

3) Using the class Pixel of #1 of this section, declare an array of class Pixel that is able to hold 499 pixels.

4) Using the class Triangle of #2 of this section, declare an array of class Triangle that is able to hold 87 triangles.

5) Using the array you declared in #3 of this section and assuming that every element in the array contains data, write the code necessary to output all data in the array.

6) Using the array you declared in #4 of this section and assuming that every element in the array contains data, write the code necessary to output all data in the array.

Section 5.2: Arrays in Functions

1) What is the output of the following code?

```
const int MAX = 10;

void InitArray(double d[], int size);
:
int main() {
    :
    double aRay[MAX];
    InitArray(aRay, MAX);
    for (int a=0; a<MAX; a++)
        cout << aRay[a] << ' ';
    :
    return 0;
}
:
void InitArray(double d[], int size)
{
    for (int x=0; x<size; x++)
        d[x] = 0.0;
}
```

2) What is the output of the following code?

```
const int MAX_NAMES = 5;

void NoName(string n[], int maxNames);
:
:
string n[MAX_NAMES];
NoName(n, MAX_NAMES);
for (int x=0; x<MAX_NAMES; x++)
    cout << n[x] << endl;
:
:
void NoName(string n[], int maxNames)
{
    for (int h=0; h<maxNames; h++)
        n[x] = "nobody";
}
```

3) What is the output of the following code?

```
struct Soil
{
    double sampleWeight;
    double clayWeight;
    double sandWeight;
    double waterWeight;
};

const int MAX_DIRT = 10000;

void InitDirt(Soil dirt[], int inUse);
    :
    :
    Soil mud[MAX_DIRT];
    InitDirt(mud, MAX_DIRT);
    for (int a=0; a<10; a++)
        cout << mud[a].sampleWeight << ' ';
    :
    :
void InitDirt(Soil dirt[], int inUse)
{
    for (int x=0; x<inUse; x++)
    {
        dirt[x].sampleWeight = 0.0;
        dirt[x].clayWeight   = 0.0;
        dirt[x].sandWeight   = 0.0;
        dirt[x].waterWeight  = 0.0;
    }
}
```

4) What is the output of the following code?

```
struct Catch
{
    double catchWeight;
    int numberFish;
};

const int MAX = 100;

void InitFish(Catch c[], int size);
    :
    :
    Catch fish[MAX];
    for (int m=0; m<10; m++)
        cout << fish[m].numberFish << endl;
    :
    :
void InitFish(Catch c[], int size)
{
    for (int x=0; x<size; x++)
    {
        c[x].catchWeight = -1.0;
        c[x].numberFish = -1;
    }
}
```

5) What is the output of the following code?

```
struct Soil
{
    double sampleWeight;
    double clayWeight;
    double sandWeight;
    double waterWeight;
};

const int MAX_DIRT = 10000;

void InitDirt(Soil dirt[], int initValue);
    :
    :
    Soil mud[MAX_DIRT];
    for (int a=0; a<MAX_DIRT; a++)
        InitDirt(mud[a], a);
    for (int a=0; a<10; a++)
        cout << mud[a].sampleWeight << ' ';
    :
    :

void InitDirt(Soil dirt, int initValue)
{
    dirt.sampleWeight = initValue;
    dirt.clayWeight   = initValue;
    dirt.sandWeight   = initValue;
    dirt.waterWeight  = initValue;
}
```

6) What is the output of the following code?

```
struct Catch
{
    double catchWeight;
    int numberFish;
};

const int MAX = 100;

void InitFish(Catch c, int initWeight, initFish);
    :
    :
    Catch fish[MAX];
    for (int m=0; m<MAX; m++)
        InitFish(fish[m], 0, m);
    for (int m=0; m<10; m++)
        cout << fish[m].numberFish << endl;
    :
    :
void InitFish(Catch c, int initWeight, initFish)
{
    c[x].catchWeight = initWeight;
    c[x].numberFish = initFish;
}
```

7) Given the following class,

```
class Pixel
{
public:
    Pixel();
    Pixel(int initRed, int initGreen, int initBlue);
    int ReturnRed();
    int ReturnGreen();
    int ReturnBlue();
private:
    int red, green, blue;
};
```

what is the output of the following code?

```
const int MAX = 100000;

void InitPixels(Pixel p[], int size);
    :
    :
    Pixel dots[MAX];
    InitPixels(dots, MAX)
    for (int x=0; x<2; x++)
        cout << dots[x].ReturnRed() << ' '
            << dots[x].ReturnGreen() << ' '
            << dots[x].ReturnBlue () << endl;
    :
    :
void InitPixels(Pixel p[], int size)
{
    for (int x=0; x<size; x++)
        p[x] = Pixel(x,x+1, x+2);
}
```

8) Given the following class,

```
class Triangle
{
public:
    Triangle();
    Triangle(int iX1, int iY1, int iX2, int iY2, int iX2, int iY2);
    void ReturnPair1(int &x, int &y);
    void ReturnPair2(int &x, int &y);
    void ReturnPair3(int &x, int &y);
    bool IsSameAs(Triangle t);
private:
    int x1, y1, x2, y2, x3, y3;
};
```

what is the output of the following code?

```
const int MAX_TRI = 300;

void InitTriangle(Triangle tAngle);
    :
    :
    Triangle t[MAX_TRI];
    int p1x, p1y, p2x, p2y, p3x, p3y;
    for (int x=0; x<MAX_TRI; x++)
        InitTriangle(t[x]);

    t[0].ReturnPair1(p1x, p1y);
    t[1].ReturnPair2(p2x, p2y);
    t[2].ReturnPair2(p3x, p3y);

    cout << '(' << p1x << ',' << p1y << ')' << endl;
    cout << '(' << p2x << ',' << p2y << ')' << endl;
    cout << '(' << p3x << ',' << p3y << ')' << endl;
    :
    :
void InitTriangle(Triangle tAngle)
{
    tAngle = Triangle(-1, -1, -2, -2, -3, -3);
}
```

Section 5.3: Programming with Arrays

1) If the file "numbers.dat" has the following values stored in it:

```
17 19 22 23 -1 34
```

What are the contents of the array *nums* after the following code is executed?

```
const int MAX = 10000;
int nums[MAX];
int count=0, n;
ifstream inFile;

inFile.open("numbers.dat");
if (!inFile)
{
    cout << "No File!";
    exit(1);
}

inFile >> n;
while (inFile && count < MAX)
{
    nums[count] = n;
    count++;
    inFile >> n;
}
```

2) If the file "names.dat" has the following values stored in it:

```
John Paul George Ringo
```

What is the output of the following code?

```
const int SIZE = 400;
string nameList[SIZE], name;
int numNames=0;
ifstream nameFile;

nameFile.open("names.dat");
if (!nameFile)
{
    cout << "No File!";
    exit(1);
}

nameFile >> name;
```

```

while(nameFile && numNames < SIZE)
{
    nameList[numNames] = name;
    numNames++;
    nameFile >> name;
}

for (int x=numNames-1; x >= 0; x--)
    cout << nameList[x] << endl;

```

3) Given that the contents of the following two arrays are both in ascending order,

```

const int MAX_ITEMS = 10000;
double a[MAX_ITEMS], b[MAX_ITEMS];

```

and given that the variables *aCount* and *bCount* contain the number of elements in the arrays *a* and *b* are in use, merge the values in arrays *a* and *b* into the following array *c* so that the values recorded in array *c* are preserved in ascending order.

```

double c[MAX_ITEMS];

```

4) Given that the contents of the files "a.dat" and "b.dat" are numbers with fractions and are in ascending order, merge the values in the two files into the following array *d* so that the values recorded in array *d* are preserved in ascending order.

```

const int MAX = 100000;
double d[MAX];

```

5) Given that the array *basketCases* contains the integer numeric grades of students in the class "Advanced Basket Weaving" and that each student has 3 grades, compute and output the average of each student. Note that the grades of student #1 are located in elements with indices 0, 1 and 2; the grades of student #2 are located in elements with indices 3, 4 and 5; etc. Output the average grade for each student next to the student's number. The variable *numStu* will contain the number of students in the actually in the class.

```

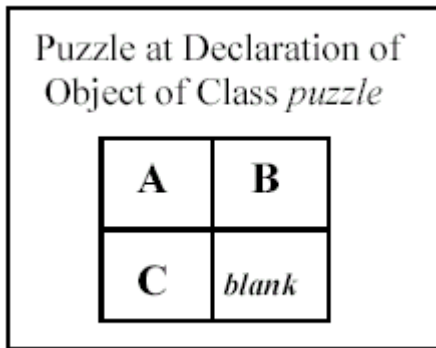
const int MAX_GRADES = 300;
int numStu;
int basketCases[MAX_GRADES];

```

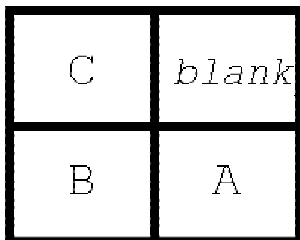
6) Given the class *Puzzle*,

```
class Puzzle
{
public:
    Puzzle();
    void MoveSquareUp(char squareToMove);
    void MoveSquareDown(char squareToMove);
    void MoveSquareLeft(char squareToMove);
    void MoveSquareRight(char squareToMove);
    void ShowPuzzle();
private:
    SquareType squareA, squareB, squareC;
};
```

and that a puzzle always starts off with the following configuration,



create an array of 100 elements of class *Puzzle*, then write the code to change the configuration of every puzzle in the array to as follows,



then show each puzzle.

7) Using the class definition listed below, write the code to create an array of 100 elements of class *LaCucaracha*. Then, write the code necessary to have each element act like the nasty bug that it is until all of them are dead. Each element should do the following:

- a) if the nasty thing is dead, do nothing
- b) if the nasty thing is alive and the light is on, the bug should be stomped at
- c) if the nasty thing is still alive after being stomped at, it should move
- d) if the nasty thing is alive and the light is off, it should eat if there is food
- e) if the nasty thing is alive and the light is off, but there is no food, it should change direction and move

Of course, when no elements are alive, the program should halt.

```
class LaCucaracha
{
public:
    LaCucaracha(); // constructor that sets object to alive
    void StompBug(); // has a 10% chance of killing the bug
    bool IsAlive(); // returns true if bug is alive, false if dead
    bool IsFood(); // returns true if there is food, false if not
    bool IsLight(); // returns true if there is light, false if not
    void Eat(); // eats the available food
    void ChangeDirection(); // changes the direction the bug
    void RunLikeHeck(); // move the bug in the direction facing
private:
    bool alive;
    int directionFacing;
    int locationX, locationY;
};
```

Programming Projects using Arrays

1) Online Sales: A Project with Arrays of Structs and Functions

Many retail businesses accept orders on-line. As the customer selects items from a list, the menu number of the object is used to examine the store inventory data base, the item's price and product description are located, counts are reduced, and a receipt is generated. Your task is to write a program that simulates this process.

Your program will need to read the inventory information from the data file "inventory.dat" from disk into an array of structs. The data in the inventory file is written one unit of data per line, beginning with its product description (a string), its price and the quantity of that item in stock. Your program will need to copy the revised version of the inventory back to the data file after all purchases have been processed.

The program generates an interactive menu of item numbers, item descriptions and their prices that are available for purchase. The customer selects one item at a time and the quantity desired of that item until the user selects to check out or cancel the purchase.

Processing customers' orders involves having the customer select products from the menu and designating a quantity for each item selected. Each selection is stored in a list in a second array of structs consisting of the menu number of the item selected and the quantity desired.

If the customer cancels the order at any time before checking out, the array containing selected item numbers has its values discarded.

If the customer elects to check out, the customer's receipt should be printed out (to the monitor). The receipt should consist of each item selected with the price of the item, the quantity selected and the extended amount (item quantity times item price) in the form of a table, with the total amount shown at the bottom of the table. In addition, the amount on hand of each item that the customer purchased should be adjusted in the inventory array. (Note: The customer may not purchase more of an item that is on hand.)

When the customer completes or cancels a purchase, the program should shut down and, if there have been changes to values in the inventory array, the values in the inventory array should be used to overwrite the file "inventory.dat".

The inventory array will contain no more than 20 items and is to be based on the following struct:

```
struct item {
    string description;
    float  price;
    int    quantity;
};
```

Hint: Use the index of each element in the inventory array as its number in the menu listing.

Example of data in the file "inventory.dat" : (Note: There are NO BLANK LINES in the data file, no matter how screwed up your printout is.) You must create your own "inventory.dat" file.

```
      :
      :
Ear Wax Remover
2.35
100
Ogre Repellant
23.99
10
Little Debbie Swiss Rolls
.27
23
      :
      :
```

The array of selected items will contain no more than 20 items and is to be based on the following struct:

```
struct selection {
    int menuNumber;
    int quantity;
};
```

The program should be decomposed into functions. You are responsible for function decomposition and the design of each function interface. No global variables are allowed. Global constants are encouraged when advantageous.

2) Multiple Candy Machines: A Project using an Array of Classes

This programming project consists of making improvements to class candy machine from the programming project Candy Machine of the workbook chapter 6 and 7.

Make the following improvements to the class candy machine.

=====

a) A candy machine object must have a name. This name will be given to the constructor during the declaration of a candy machine object. You will need a default constructor and a constructor that accepts this name.

b) In the original candy machine class, there were a fixed number of items for sale, all of the same price. In this program, there will be a flexible number of items that can be placed on sale, from 0 to 20. Each item will have a name, a quantity and a price. The information about an item is to be stored in a struct. The information about all items is to be stored in an array of these structs. The constructor will read this information from a file called "machine-name.dat", where machine name is the name given to the candy machine at the time of the declaration of the candy machine object. The destructor will return all of this information to the same file. The menu, choice and purchasing system must be rewritten to accommodate this change. HINT: USE THE INDEX NUMBER OF THE ITEM'S STRUCT IN THE ARRAY AS THE CHOICE NUMBER TO BE ENTERED BY THE USER.

c) The data file created by each machine must be in the following form:

```
password
total money
itemQty      itemPrice      itemName
itemQty      itemPrice      itemName
itemQty      itemPrice      itemName
:            :            :
:            :            :
```

For example:

```
gummo
53.50
15  .35  Strawberry Potato Chips
10  1.00  Ham Sandwich
3   1.75  Dental Floss
19  .25  Spam Crackers
8   .50  Swiss Rolls
1   1.00  Tic-Tacs
```

Make the following changes to the main function.

=====

a) Declare an array of 4 Candy machine class objects.

b) Using calls to the candy machine class constructor, set the machine name in each machine. The names of the candy machines must be "Chico", "Harpo", "Groucho" and "Zeppo". For example:

```
Candy vend[4];
vend[0] = Candy("Chico");
vend[1] = Candy("Harpo");
      :
      :
```

c) Have the loop in the main function display a menu of candy machines. The program should allow the user to choose a machine to make a purchase from or to halt the program. If the user chooses to make a purchase from a machine, the program should call the member functions from the appropriate machine. If the user chooses to halt the program, the loop in the main function should stop and the program should terminate. (Note: In order for this program to function properly, the candy machine class must have a destructor that stores state information to the correct file. This version of the candy machine class does not need an option to shut down as this option is being covered in main.)

Final Instructions

=====

Name your source files 'candyarray.cc', 'machineb.h' and 'machinab.cc'. Compile your program and test it. (see Some Unix Help for quick assistance)