

# CMPS 150 Workbook: Chapters 6 and 7

## Section 6.1: Structures

1) Declare a variable of the following struct and assign appropriate values to each of the data members.

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

2) Declare a variable of the following struct and assign appropriate values to each of the data members.

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

3) Declare a variable of the struct *Dinosaur* from #1 and write the code necessary for the user to enter values into each data member.

4) Declare a variable of the struct *Phone* from #2 and write the code necessary for the user to enter values into each data member.

5) Using the variable of struct *Dinosaur* declared in #3, output the values stored in all data members to the monitor screen.

6) Using the variable of struct *Phone* declared in #4, output the values stored in all data members to the monitor screen.

7) What is the output of the following code? (See #1 for the definition of struct *Dinosaur*.)

```
Dinosaur barney, dino;
barney.name="T-Rex";
dino = barney;
cout << dino.name << endl;
```

8) What is the output of the following code? (See #2 for the definition of struct *Phone*.)

```
Phone maBell, gte;
maBell.phoneNumber = "1234567890";
gte = maBell;
cout << gte.phoneNumber << endl;
```

9) Write a function that accepts two arguments of struct *Dinosaur* (see #1) and determines if the two structs contain information about the same type of dinosaur. The function should return true if the dinosaur names match, but should return false if they do not match.

10) Write a function that accepts two arguments of struct *Phone* (see #2) and determines if the two structs contain the same phone number. The function should return true if the phone numbers match, but should return false if they do not match.

11) Create a struct that contains data members to represent information about a student in a class. The struct should contain data members for the student's name, id and numeric grades on 3 exams.

12) Create a struct that contains data members to represent information about a house that is for sale. The struct should contain data members for the location of the house, owner of the house, number of bedrooms, total square feet in the house and asking price for the house.

13) Create a struct that contains data members to represent information about a truck in a fleet of trucks. The struct should contain data members for the truck ID number, year purchased, license number and vehicle identification code.

14) Create a struct that contains data members to represent information about a purchase by customers from a store that sells medicinal leeches. The struct should contain data members for the customer number, date of the purchase, method of payment, number of *hirudo medicinalis* ordered, number of *hirudinaria manillensis*, and number of *haementeria ghilianii* ordered.

## Section 6.2 and 7.1: Classes and Constructors

1) Using the class definition listed below, write the code to create an object of class *Puzzle*. Then, write calls to the appropriate member functions to move square A to the lower right corner so that the object ends up with configuration of:

C	<i>blank</i>
B	A

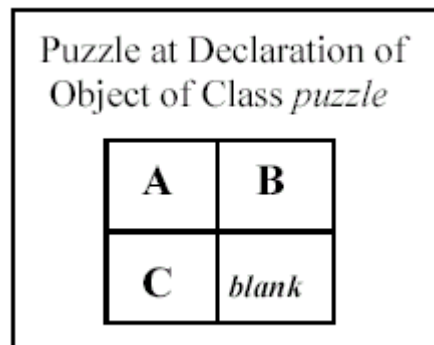
Game Rules:

Square may only be moved to an adjoining area.

Squares may only be moved to a blank area.

Moving a square leaves a blank area.

Squares are identified to the functions by passing the letter name of the square.



```
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;
};
```

2) Using the definition of class *Puzzle* from #1, define an object of class *Puzzle* and write the calls to member functions that end with the *Puzzle* object in the configuration:

B	C
blank	A

3) Using the class definition listed below, write the code to create an object of class *LaCucaracha*. Then, write a call to the appropriate member functions to stomp the bug. If the bug survives the stomping, have the bug change direction and run in the new direction it is facing.

```
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 of the bug
    void RunLikeHeck(); // move the bug in the direction facing
private:
    bool alive;
    int directionFacing;
    int locationX, locationY;
};
```

4) Using the class *LaCucaracha* from #3, create an object of class *LaCucaracha* and write calls to its member functions so that the bug runs if the light is on, otherwise, if the light is off and there is food, it eats. In either case, the bug must be alive in order to do any of this.

5) Using the class *LaCucaracha* from #3, create an object of class *LaCucaracha* and write the code necessary to have it act like the nasty bug that it is until it is dead. This code should do the following:

- a. if the nasty thing is dead, halt
- 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

6) Rewrite #5 so that there are two objects of class *LaCucaracha*. The code should continue until both bugs are dead.

7) Define, but do not implement, a class called *Employee*. This class should store the following data:

- employee name
- employee id
- employee hourly rate of pay
- employee hours worked in week

This class should have constructors such that:

- one constructor has no arguments
- another constructor receives name and id via arguments and stores the name and id

This class should have member functions to:

- receive (as arguments) and store hourly rate of pay
- receive (as arguments) and store hours worked
- return the gross pay

8) Define, but do not implement a class called Student. This class should store the following data:

- name
- id
- year in school

The class should have constructors such that:

- one constructor has no arguments
- one constructor receives (as arguments) and stores the name, id and year in school

The class should have member functions to:

- add one to the year in school
- return the name
- return the id
- return the year in school

9) Implement the constructors and member functions of your class design from #7.

10) Implement the constructors and member functions of your class design from #8.

## Programming Project: Candy Machine

Design and implement a class for a candy vending machine according to the specifications below and use an object of the class in a program.

### Candy Machine Class Specifications

=====

a) The candy machine class stocks and sells four items:

- potato chips
- jawbreakers
- chocolate chip cookies
- vanilla chewing gum.

The candy machine class must store information about each of these products. This information includes the quantity on hand and the price of each type of item. In addition, a password is stored in order for the owner / stocker to gain special access to the machine.

b) The candy machine class also keeps track of the amount of money collected.

c) When a candy machine object is initialized for the first time, the quantities for each item are to be set at 20 units for each item, there is no money in the machine, there are no sales that have been made and the password is set to "password". After the first run of a candy machine object, the data for the quantity of each item, the amount of money on hand and the current password will be read from a file called "machine.dat".

d) 20 units is to be considered the maximum storage available for the candy machine for an item.

e) When a candy machine object passes from scope or the program ends, the data stored in the object is written to a file called "pa8.dat".

f) Each item costs 50 cents.

g) In addition to the above, the candy machine class must also have member functions to output a menu of available items, the cost of the items, record the transactions (i.e. sum the money and decrement the amount available of the item selected), accept the users money and give change. The menu function may not receive any input nor call any other functions. The function that accepts user choices may not perform any processing but may call other member functions to do so.

- h) There must be an unlisted choice on the menu that the owner / stocker of the candy machine object can access that will return the amount of money in the system if the user inputs a specified password. There must also be a means of setting this password separate from constructors. All constructors should set the password to "password" initially. The user should only be able to change the password if the password is known.
- i) There must be an unlisted choice on the menu that allows the owner / stocker of the candy machine object to restock the machine. When a machine is restocked, items are set to the maximum allowed and all money is removed from the machine. The password must be entered in order to be able to open the machine and restock it.
- j) There must be an unlisted choice on the menu that allows the owner / stocker of the candy machine object to shut the machine down. The member function that receives the user's choice must return a value indicating that it is time to shut the machine down. The password just be entered in order to shut the machine down.
- k) A candy machine object cannot sell items that are out of stock.
- l) If there is no money to give change, return the users money and make no sale.

#### Specifications for Program to use Candy Machine Class

=====

Write program that will operate an object of the candy machine class. The program is to run the machine until the owner / stocker elects to shut the machine down. There is to be no I/O in the program. All I/O is to be performed by calls to member functions of the object of class Candy Machine.

In addition, name your source files 'candy.cc', 'machine.h' (class definition file) and 'machine.cc' (class implementation file).

Compile your program and test it.