**CS-255 Exam 1 Study Guide**

**Parallel Arrays**

* Sorting:
  + Selection Sort

Void selectionSort(string names[], int studentID[], int num)

{

string tempName;

int temp = 0, minIndex = 0;

for(int i = 0; I < num; i++)

{

minIndex = i;

for(int j = i+1; j < num; j++)

{

if(names[j] < names[minIndex])

{

minIndex = j;

}

}

tempName = names[i];

temp = studentID[i];

names[i] = names[minIndex];

studentID[i] = studentID[minIndex];

names[minIndex] = tempName;

studentID[minIndex] = temp;

}

}

**Structs**

Data structure that has a group of related items that can consist of different types

* Creating Structs:
  + Struct Example

struct StudentType

{

string firstName; //member names & types

string lastName;

int age;

double gradePointAverage;

};

* Passing a struct to a function
  + Example

void print (StructureName objectVariable)

{

Cout << objectVariable.name << endl;

Cout << objectVariable.color << endl;

}

* Creating Structure Variable (object)
  + At top of main, StructureName objectVariable;
* Array of structs

StructureName object[MAX\_SPOTS];

* Sorting array of structs

void sortingFunction(StructureName array[], int num)

{

StructureName temp;

Int minIndex = 0;

for(int i = 0; i < num; i++)

{

minIndex = i;

for(int j = i+1; j < num; j++)

{

If(array[j].age < array[minIndex].age)

{

minIndex = j;

}

}

temp = array[i]

array[i] = array[minIndex]

array[minIndex] = temp;

}

}

* Accessing Members of Struct

structName.member = “BLAH”;

**Object Oriented Programming**

* 3 Principles to OOP:
  + Encapsulation: the ability to combine data and operations on that data in a single unit
  + Inheritance: the ability to create new objects from existing objects
  + Polymorphism: the ability to use the same expression to denote different operations
* Classes are fundamental entities of OOP

**Classes**

* Creating objects

Class AutoMobile

{

private:

string make;

string model;

double year;

double milage;

public:

//SETTERS AND GETTERS

void setMake(string);

void setModel(string);

void setYear(int);

void setMilage(int);

string getMake() const;

string getModel() const;

int getYear () const;

int getMilage () const;

AutoMobile(); //constructor AUTOMATICALLY VOID

//pass values to constructor

AutoMobile(string, string, int, int);

~AutoMobile(); //destructor used for data clean up

setAutoMobile(string, string, int, int);

}

* Defining objects
  + AutoMobile car1, car2;
* Passing objects to a function
  + In order to include objects and members of a class, we must use the :: within the function
  + Example: If I wanted to print any set members of a class

Void AutoMobile::print() const

{

Cout << make << “ “ << model << “ “ << year << “ “ << milage << endl;

}

//NOTE: Within the main function, calling the print function //would look like car1.print(); after everything was set and got

* Array of objects

AutoMobile cars[MAX\_CARS];

* Reading data from file into objects

Void readAutoMobilesFromFile(AutoMobile car[], int &numCars)

{

numCars = 0;

ifstream carFile;

string make = “”;

string model = “”

int year = 0, milage = 0;

carFile.open(“cars.txt”);

getline(carFile, make);

while(!carFile.eof())

{

Getline(carFile, model)

carFile >> ws; //AFTER GETLINE AND BEFORE INPUT

carFile >> year;

carFile >> milage;

//THIS WILL COME FROM SEPARATE FUNCTION

car[numCars].setAutoMobile(make, model, year, milage);

numCars++;

getline(carFile, make);

}

carFile.close();

}

//SEPARATE FUNCTION

Void AutoMobile::setAutoMobile(string setMake, string setModel, int setYear, int setMilage)

{

make = setMake;

model = setModel;

year = setYear;

milage = setMilage;

}

* Setters and Getters
  + Setters: You can set any value of the member within the main function or though user created functions
  + Ex:

Void setMake(string); //this will be inside the class

Void AutoMobile::setMake(string inMake)

{

Make = setMake;

}

* + Getters: Returns inputted value into the class member which can be accessed by user (but not for modification)
  + Ex:

String AutoMobile::getMake() const

{

Return make;

}

* Constructor: used to initialize an object
  + Is automatically void
  + You can use a second constructor to pass along default values to an object
  + Automatically called at main function
  + Ex:

AutoMobile();

AutoMobile::AutoMobile()

{

Make = “”;

Model = “”;

Year = 0;

Milage = 0;

}

//Second Constructor

AutoMobile::AutoMobile(string inMake, string inModel, int inYear, int inMilage)

{

Make = inMake;

Model = inModel;

Year = inYear;

Milage = inMilage;

}

* Destructor: called when an object goes out of scope
  + Cleans up activities to free up memory
  + Ex:

~AutoMobile();

AutoMobile::~Automobile()

{

Cout << “This is only to show that the destructor worked”;

}

**UML DIAGRAMS**

Ex:

|  |
| --- |
| AutoMobile |
| * Make: string * Model: string * Year: int * Milage: int |
| setMake(string): void  setModel(string): void  setYear(int): void  setMilage(int): void  getMake() const: string  getModel() const: string  getYear () const: int  getMilage () const: int |

**Creating a project with multiple files**

* Things needed in a project:
  + Class Header
    - Ex: AutoMobile.h
  + Class Implementation file
    - Ex: AutoMobile.cpp
    - Must include class header
  + Main Driver
    - Ex: main.cpp
    - Will also include class header
* When complied, it will compile the implementation and the main driver into object code
  + Ex: main.o

**Abstract Data Type**

* Separates the logical properties from the implementation details
* ADT consists of:
  + Name (INT)
  + Domain – values belonging to ADT (123456789)
  + Operations on the data (+ - % / \*)

**Static Variables**

* Exists between function calls
* Classes can have static members, functions, and variables
* Ex:

Void countFunc()

{

Int a = 0;

//Value doesn’t restart every time we return to function

Static int b = 0;

A++;

B++;

Cout << “a is “ << a << endl << “b is “ << b << endl;

}

Int main()

{

countFunc();

coutFunc();

countFunc();

return 0;

}

//A will reset every time, but B will not

**Classes with static members, static functions, and static variables**

* Ex:

Class Illustrate

{

Private:

Int x;

Static int y;

Public:

Static int count;

Void print() const;

Void setX (int a);

Static void increment();

Illustrate(int a = 0);

};

* With static functions, you can call functions using a class without using the objects