**Data Structures (H16Y 35) – Class Exercise**

# What do you understand by the term ‘Arithmetic Overflow’?

In a digital computer, the condition that occurs when a calculation produces a result that is greater than a given register or storage location can store or represent

# What are ‘Control Structures’?

A control structure is a block of programming that analyses variables and chooses a direction in which to go based on given parameters. The term flow control details the direction the program takes (which way program control "flows")

## State types of Control Structures used in C# programming

Flow of control through any given function is implemented with three basic types of control structures:

**Sequential**: default mode. Sequential execution of code statements (one line after another) -- like following a recipe

**Selection**: used for decisions, branching -- choosing between 2 or more alternative paths. In C++, these are the types of selection statements:

* if
* if/else
* switch

**Repetition**: used for looping, i.e. repeating a piece of code multiple times in a row. In C++, there are three types of loops:

* while
* do/while
* for

# What are ‘Floating point numbers’?

The term floating point refers to the fact that a number's radix point (decimal point, or, more commonly in computers, binary point) can "float"; that is, it can be placed anywhere relative to the significant digits of the number.

## Give examples of **Floating point numbers** commonly used in C# programming

* Float
  + float num = 14f;
  + Precision: 7 digit
* Double
  + double num = 200;
  + Precision: 16 digit

# What is the function of the ‘GetType ()’ syntax used in DS algorithm in C# programming?

GetType gets the runtime type of an instance.

Used to compare data types.

# Differentiate between ‘Console.WriteLine’ and ‘Console.ReadLine’, what will you apply each of the syntax for during programming?

Console.WriteLine(“XXX”);

This will write XXX to the console or whatever else you put between the brackets. This could be used to debug the program.

Console.ReadLine();

This will read the a line from the standard input stream(keyboard). This can be used to get input from the user.

# Define Arrays

An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

## Name types of arrays commonly used in C# programming

Arrays can be divided into the following four categories.

### Single-dimensional arrays

Single-dimensional arrays are the simplest form of arrays. These types of arrays are used to store number of items of a predefined type. All items in a single dimension array are stored contiguously starting from 0 to the size of the array -1.

double[] doubleArray = new double[5];

char[] charArray = new char[5];

bool[] boolArray = new bool[2];

string[] stringArray = new string[10];

### Multidimensional arrays or rectangular arrays

A multi-dimensional array, also known as a rectangular array is an array with more than one dimension. The form of a multi-dimensional array is a matrix.

int[,] numbers = new int[3, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 } };

string[,] names = new string[2, 2] { { "Rosy", "Amy" }, { "Peter", "Albert" } };

### Jagged arrays

Jagged arrays are arrays of arrays. The elements of a jagged array are other arrays.

**One way of initializing:**

jaggedArray[0] = new int[] { 1, 3, 5, 7, 9 };

jaggedArray[1] = new int[] { 0, 2, 4, 6 };

**The other:**

jaggedArray[2] = new int[] { 11, 22 };

int[][] jaggedArray2 = new int[][]

{

new int[] {1,3,5,7,9},

new int[] {0,2,4,6},

new int[] {11,22}

};

# Produce one example each of algorithm used for the common array types mentioned in Question 6(ii) in C# programming.

## Single Dimensional

**min = a[0]; // Assume a[0] is the minimum**

**for all elements a[1], a[2], ... of array a do**

**{**

**if ( a[i] < min )**

**{**

**min = a[i]; // We found a smaller minimum**

**}**

**}**

**print min;**

## Multi-Dimensional

for(int i = start[0], countx = 0, count < xcount; countx++, i++) {

for( int j = start[1], county = 0; county < ycount; county++, j++) {

//logic here

}

}

# What do you understand by ‘*Type Casting*’?

Doing a cast in C# is telling the compiler to do an explicit conversion to convert the type of an object from one to another, and by explicit it means that you are aware that data may be truncated during the operation. For example : converting a decimal to a float or a double to int

## State two types of Type casting commonly used in C# programming

**Implicit conversions**: No special syntax is required because the conversion is type safe and no data will be lost. Examples include conversions from smaller to larger integral types, and conversions from derived classes to base classes.

**Explicit conversions** (casts): Explicit conversions require a cast operator. Casting is required when information might be lost in the conversion, or when the conversion might not succeed for other reasons. Typical examples include numeric conversion to a type that has less precision or a smaller range, and conversion of a base-class instance to a derived class.

## Produce an example of algorithm in C# programming used to perform Type casting operation for each type mentioned in Question 9(ii).

### Implicit

// Implicit conversion. num long can

// hold any value an int can hold, and more!

int num = 2147483647;

long bigNum = num;

### Explicit

class Test

{

static void Main()

{

double x = 1234.7;

int a;

// Cast double to int.

a = (int)x;

System.Console.WriteLine(a);

}

}

// Output: 1234

# What are Abstract Data Type Structures?

In computer science, an abstract data type (ADT) is a mathematical model for data types, where a data type is defined by its behaviour (semantics) from the point of view of a user of the data, specifically in terms of possible values, possible operations on data of this type, and the behaviour of these operations.

## List 5 components of ADT structures

* A name (the name of the ADT)
* The data type or the data types represented by a collection of data, which is structured in some manner [each type has been previously defined in a ADT (or it is a primitive type)]
* Functions that operate on the data [the functions will specify the domain (parameters) and range (return type)]
* Preconditions for each of the functions (not shown in the example)
* Post-conditions for each of the functions (not shown in the example)
* Error conditions that we need to consider.

## State the differences between Stack & Queue ADTs.

In very simple terms, a stack is a collection of objects in which objects are accessed in LIFO (Last In First Out) fashion. Whereas a queue is a collection of objects in which objects are accessed in FIFO (First In First Out) sequence.

# Differentiate between ‘Simple Data Structures’ and ‘Structured Data’

Simple data represents the individual pieces of information contained in a data set e.g. Harry, 120.10 and 10.

Structured data applies to groups (or collections) of data items that are linked together (i.e. related) for some purpose.

## Give 4 examples each for the Data structure types in Question 12(i).

Examples of Structured Data

* Arrays
* Records
* Tables (Hash Tables)
* Files

Examples of simple data:

* **Numeric**: There are two main numeric data types: Integer and Real
  + **Integers** - Positive and negative whole numbers, i.e. numbers without a decimal point, e.g. 12, 0, -24.
  + **Real or Floating Point** - Numbers with fractional parts, i.e. with a decimal point, e.g. 12.4, -0.6, 3.149
* **Character**: These data type can be thought of in terms of two sub-categories:
  + **Single Characters**: could represent a number, an alphabetic character, a punctuation mark, mathematical symbol, or a special symbol, e.g. '+', 'A' and '£‘
  + **Strings.**
* **Logical (Boolean)**: Data type characterised by variables that can only take values *True* and *False*

# What is the difference between Explicit & Implicit Shortening Conversions?

**Implicit conversions**: No special syntax is required because the conversion is type safe and no data will be lost. Examples include conversions from smaller to larger integral types, and conversions from derived classes to base classes.

**Explicit conversions** (casts): Explicit conversions require a cast operator. Casting is required when information might be lost in the conversion, or when the conversion might not succeed for other reasons. Typical examples include numeric conversion to a type that has less precision or a smaller range, and conversion of a base-class instance to a derived class.

# Define SMA and DMA

**Static Allocation** means, that the memory for your variables is allocated when the program starts. The size is fixed when the program is created. It applies to global variables, file scope variables, and variables qualified with static defined inside functions.

**Dynamic memory allocation** is a bit different. You now control the exact size and the lifetime of these memory locations. If you don't free it, you'll run into memory leaks, which may cause your application to crash, since it, at some point cannot allocation more memory.

## State the differences between SMA & DMA

In general, static memory allocation is the allocation of memory at compile time, before the associated program is executed, unlike dynamic memory allocation or automatic memory allocation where memory is allocated as required at run time.

## State four functions used in C# to allocate memory during programming.

malloc();

calloc();

realloc();

free();