COSC 120 Review

- Array
- Elementary Sorting Algorithms
  - Selection Sort
  - Insertion Sort
  - Bubble Sort
- Structured Data Type
- Structured Data Type Union

Review for COSC 120
(Array Declaration)

- An array is a series of elements of the same type placed in contiguous memory locations.
- Like a regular variable, an array must be declared before it is used.
- A typical declaration for an array in C++ is:

```
type Array_Name [number_of_element];
```


Review for COSC 120
(Accessing Array Element with Index)

```
int Day [5];

Day[0] = 10;
Day[1] = 20;
Day[3] = Day[0] + Day[1];
Day[4] = Day[1] + Day[2];
```


Review for COSC 120
(Initializing elements of an Array)

- When declaring a regular array of local scope (within a function, for example), if we do not specify otherwise, its elements will not be initialized to any value by default, so their content will be undetermined until we store some value in them.
- The elements of **global** and **static arrays**, on the other hand, are automatically initialized with their default values, which for all fundamental types this means they are filled with zeros.

Review for COSC 120
(Initializing elements of an Array)

```
// global-local.cpp
// Array with default initialization or not
// But it causes unsafe memory access
#include <iostream>
using namespace std;

int Global[10];

int main()
{
        int Local[10];
        static int Static[10];


        for (int i = 0; i <10; i++)
                cout << " Global["<<i<<"] =" << Global[i]<<endl;
        for (int i = 0; i <10; i++)
                cout << " Local["<<i<<"] =" << Local[i]<<endl;
        for (int i = 0; i <10; i++)
                cout << " Static["<<i<<"] =" << Static[i]<<endl;


        return 0;
}
```


Review for COSC 120
(No Array Bounds Checking in C++)

- C++ gives the freedom to programmers to work with computer memory.
- But, sometimes it causes unsafe memory accessing.
- If a programmer try to modify out of array bound, it might overload a space which is used by other part of program!!

Review for COSC 120
(No Array Bounds Checking in C++)

```
// C++ does not array bound checking.
// But it causes unsafe memory access
#include <iostream>
using namespace std;

int main()
{
        int day[3];

        for (int i = 0; i <10; i++)//might be segmentation fault
                day[i] =20;
        for (int i = 0; i <10; i++)
                cout << " day["<<i<<"] =" << day[i]<<endl;
        return 0;
}
```


Review for COSC 120
(Copy Elements of An Array to Another)

To copy elements of an array to another array, each element of an array must copy to another array.

```
// element of array cannot copy with simple assignment
#include <iostream>
using namespace std;

int main()
{
        int array1[4]={0,1,2,3};
        int array2[4];

        array2 = array1;// might be invalid or dangling memory space
        for (int i = 0; i < 9; i++)
                cout << "array2 [" << i <<"] = "<< array2[i]<<endl;
        return 0;
}
```


Review for COSC 120
(Copy Elements of An Array to Another)

```
// element of array cannot copy with simple assignment
#include <iostream>
using namespace std;

int main()
{
        int array1[4]={0, 1, 2, 3};
        int array2[4];

        for (int i = 0; i < 4; i++)
                array2[i]=array1[i];

        for (int i = 0; i < 4; i++)
        {
                cout << "array1 [" << i <<"] = "<< array1[i];
                cout << ", array2 [" << i <<"] = "<< array2[i]<<endl;
        }
        return 0;
}
```


Review for COSC 120
(Summing the Values in a Numeric Array)

```
// Summing the values in a numeric Array
#include <iostream>
using namespace std;

int main()
{
        const int size = 5;
        int sum = 0;
        int array[5] = {12, 34, 25, 56, 14};
        for (int i = 0; i < size; i++)
                sum += array[i];
        cout << "The sum of array =" <<sum <<endl;
        return 0;
}
```


Review for COSC 120
(Find a Maximum Element in a Array)

```
// Find a Maximum element in a Array
#include <iostream>
using namespace std;

int main()
{
     const int size = 5;
     int array[size] = {12, 34, 25, 56, 14};
     int MaxIndex =0;
     for (int i = 1; i < size; i++)
     {
             if (array[i]> array[MaxIndex])
                 MaxIndex = i;
     }
     cout << "The Maximum element of array = " <<array[MaxIndex] <<endl;

     return 0;
}
```


Review for COSC 120
(Find a Minumum Element in a Array)

```
// Find a Minumum element in a Array
#include <iostream>
using namespace std;

int main()
{
     const int size = 5;
     int array[size] = {12, 34, 25, 56, 14};
     int MinIndex =0;
     for (int i = 1; i < size; i++)
     {
             if (array[i]< array[MinIndex])
                 MinIndex = i;
     }
     cout << "The Minumum element of array = " <<array[MinIndex] <<endl;

     return 0;
}
```

Review for COSC 120
(Arrays as Function Arguments)

- C++ support call by value and call by reference.
- Since there are usually more than one element in an array, array can only pass parameter as reference.

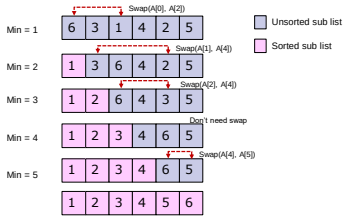
Review for COSC 120
(Initializing with Strings)

```
//Charecter string initialization with an array of charecter
#include <iostream>
using namespace std;

int main()
{
    char HisName[20] ="Michael Jackson";
    char YourName[]={'C','h','r','i','s','t','i','n','a','
','A','g','u','i','l','e','r','a','\0'};
    char HerName[] = {'B','r','i','t','n','e','y','
','S','p','e','a','r','s'}; //depends on system
    cout << "His Name is "<< HisName<<endl;
    cout <<"Your Name is " <<YourName <<endl;
    cout << "Her name is " << HerName <<endl;
    return 0;
}
```


Review for COSC 120
(Selection Sort)

- Input: A sequence of n numbers $<a_1, a_2, a_3, \ldots a_n>$
- Output: Permutation $<a'_1, a'_2, a'_3, \ldots a'_n>$ of input sequence such that $a'_1 \le a'_2 \le a'_3 \le \ldots \le a'_n$



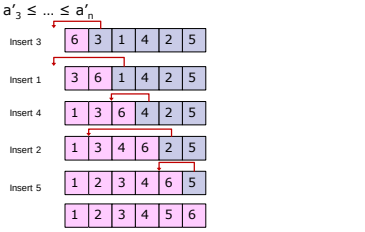
Review for COSC 120
(Selection Sort)

- For each iteration, a minimum is found in the sub list and it is placed into right position in sorted sub list.

```
1       Selection sort (A)
2       {
3          for j = 0 |A|-1 do
4            {
5                   Min = j
6                   for k= j+1 to |A|-1
7                   {
8                          if A{k} <A[Min] then
9                               Min = k
10                  }
11                  If Min ≠ j
12                        Swap (A[j], A[Min])
13           }
14      }
```


Review for COSC 120
(Insertion Sort)

- Input: A sequence of n numbers $<a_1, a_2, a_3, \ldots a_n>$
- Output: Permutation $<a'_1, a'_2, a'_3, \ldots a'_n>$ of input sequence such that $a'_1 \le a'_2 \le a'_3 \le \ldots \le a'_n$



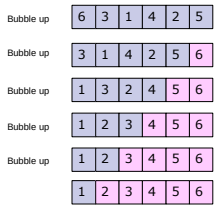
Review for COSC 120
(Insertion Sort)

- For each iteration, a element is placed into right position in sorted sub list.

```
1       Insertion sort (A)
2       {
3          for j = 2 to length of array A do
4            {
5                   key = A[j]
6                   i = j - 1
7                   while i > 0 and A[i] >key do
8                  {
9                         A[i +1] = A[i]
10                         i = i -1
11                  }
12                  A[i +1] = key
13           }
14      }
```

Review for COSC 120
(Bubble Sort)

- Input: A sequence of n numbers $<a_1, a_2, a_3, \ldots a_n>$
- Output: Permutation $<a'_1, a'_2, a'_3, \ldots a'_n>$ of input sequence such that $a'_1 \le a'_2 \le a'_3 \le \ldots \le a'_n$



Review for COSC 120
(Bubble Sort)

- For each iteration, larger element move up (or smaller element move down).

```
1       BubbleSort(A)
2       {
3          for j = |A|-1 down to 0 do
4            {
5                   for k=2 to j do
6                   {
7                         if A[k-1] > A[k]
8                               Swap (A[k-1], A[k])
9                    }
10              }
11      }
```


Review for COSC 120
(Linear Search)

```
//Liner Search returns index of a specific element in a array

int LinearSearch(int list[], int numElems, int value)
{
   int index = 0;       // Used as a subscript to search array
   int position = -1;   // To record position of search value
   bool found = false;  // Flag to indicate if the value was found

   while (index < numElems && !found)
   {
      if (list[index] == value)  // If the value is found
      {
         found = true;           // Set the flag
         position = index;       // Record the value's subscript
      }
      index++;                   // Go to the next element
   }
   return position;              // Return the position, or -1
}
```


Review for COSC 120
(Binary Search)

```
int binarySearch(int array[], int numelems, int value)
{
   int first = 0,             // First array element
       last = numelems - 1,   // Last array element
       middle,                // Mid point of search
       position = -1;         // Position of search value
   bool found = false;        // Flag
   while (!found && first <= last)
   {
      middle = (first + last) / 2;     // Calculate mid point
      if (array[middle] == value)      // If value is found at mid
      {
         found = true;
         position = middle;
      }
      else if (array[middle] > value)  // If value is in lower half
         last = middle - 1;
      else
         first = middle + 1;           // If value is in upper half
   }
   return position;
}
```


Review for COSC 120
(Writing Character String Handling Functions)

- void stringCopy(char [], char[])
- void stringCopyn(char[], char[], int)
- bool stringCompare(char[], char[])
- int stringCount(char[])

Review for COSC 120
The C++ string Class)

What is string class?

- The string class is an abstract data type.
- This is not a built-in, primitive data type but a programmer-defined data type.
- A string class is one of the C++ Standard Template Library (STL).
- In order to use the string class you should include <string> header at the begging of a program.
- It provides not only space for saving a string but also many functions to handle a string.

Review for COSC 120
(Comparing string Objects)

- The == operator cannot be used to compare two character strings defined as array of character.
- Two array of character can be compared by using the **strcmp** function or programmer defined function.
- But two string object can be compared with relational operators >, >=, <, <=, == and !=.

Review for COSC 120
(Comparing string Objects)

```
// a string class example
#include <iostream>
#include <string>
using namespace std;

void main()
{
        string yourName = "Mary";
        string myName ="Mary";
        if (yourName == myName)
                cout << "We have same name"<<endl;
        else
                cout <<"We have different name"<<endl;

}
```


Review for COSC 120
(The string Class operators)

- The string class has overloading operators for handling string objects.

Overloaded operators for string class objects
>>
<<
=
+
+=
<, <=, >, >=, ==, !=

Review for COSC 120
(Structured Data Types)

- Instead of keeping a date data in individual variables, variables can be grouped together into an ADT.

```
// Declarations for keeping a date
int Month;
int Day;
int Year;
```

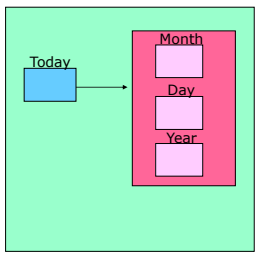
```
// a possible ADT for date
struct Date
{
        int Month;
        int Day;
        int Year;
}

Date Today;
```


Review for COSC 120
(Memory Allocation of Structured Data Type)

```
struct Date
{
        int Month;
        int Day;
        int Year;
}

Date Today;
```



Review for COSC 120
(A SDT as a Member in a SDT)

```
struct Date{
        int Month;
        int Day;
        int Year;
}; // ADT for Date

struct Student{
        char LastName[20];
        char FirstName[20];
        int IDNumber;
        int MajorCode;
        Date BirthDay; // Date ADT
};// ADT for saving a student information
```

Review for COSC 120
(Size of a Structured Data Type)

- We can check the size of structured data type by using sizeof function.

```
// size of ADT can be checked by using sizeof function
#include <iostream>
using namespace std;

struct Date{
        int Month;
        int Day;
        int Year;
};
int main ()
{
        Date Today;
        cout << " size of date is "<< sizeof(Today)<<endl;
        return 0;
}
```


Review for COSC 120
(Accessing a Members in a Structure Data)

- C++ provide dot operator to access the individual members of a structure.

```
// a possible ADT for date
struct Date
{
       int Month;
       int Day;
       int Year;
}

Date Today;
Today.Month = 11;
Today.Day = 26;
Today.Year = 2007;
```


Review for COSC 120
(Passing as a Parameter)

- Since the name of a instance of a structured data type save the location where the structured data is located in a memory, we simply pass name of instance as a parameter.

```
Student Me;
…
printStructure (Me);
```


Review for COSC 120
(Initializing a Structured Data Type)

```
//Initializing a structured data
#include <iostream>
using namespace std;

struct Date{
            int Month;
            int Day;
            int Year;
}; // ADT for Date

struct Student{
            char LastName[20];
            char FirstName[20];
            int IDNumber;
            int MajorCode;
            Date BirthDay;
};// ADT for saving a student information

void printStructure(Student);
int main ()
{
            Student Me ={"Park", "Sang-Eon", 12345, 123, {8, 27, 1968}};
            printStructure (Me);
            return 0;
}
```


Review for COSC 120
(Array of an Structured Data Type)

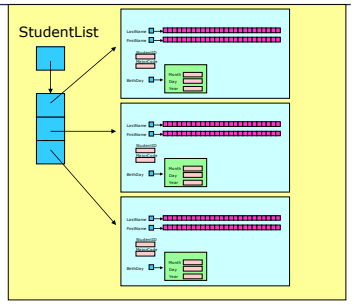
```
// Once we define an ADT, we also can build an array of any ADT
struct Date{
       int Month;
       int Day;
       int Year;
}; // ADT for a date

struct Student{
       char LastName[20];
       char FirstName[20];
       int StudentID;
       int MajorCode;
       Date BirthDay;
}; // ADT for student data

Student StudentList[3];// Array of ADT Student
```


Review for COSC 120
(Array of an Structured Data Type)

Student StudentList[3];



Review for COSC 120
(Assignment Operator Between Structured Data Types)

- Assignment operator = can be used between two same structured data type.
- An assignment operator = copy content of a structured data type into another, since system create default copy constructor.

Review for COSC 120
(Assignment Operator Between Structured Data Types)

```
//assignment operator between same instances of a sturctured data type
#include <iostream>
using namespace std;

struct Date{
        int Month;
        int Day;
        int Year;
}; // ADT for Date
int main()
{
        Date Today ={11, 28, 2007};
        Date Copy;

        Copy = Today;
        cout << "Today is " << Today.Month << "/"<<Today.Day;
        cout <<"/"<<Today.Year<<endl;
        cout << "Copy Day is " << Copy.Month << "/"<<Copy.Day;
        cout <<"/"<<Copy.Year<<endl;
        return 0;
}
```


Review for COSC 120
(Structured Data Type : union)

- A union is just like a structure data type struc but, the significant difference is that all the members of a union use the same memory area, so **only** one member can be used at a time.
- The entire variable will only take up as much memory as the largest member.

Review for COSC 120
(Structured Data Type : union)

```
union PaySource
{
        short hours;
        float sales;
};
PaySource Employee1;
```

hour = short = 2 bytes;

sales = float = 4 bytes;