CECS 282
EXAMINATION 1 REVIEW

Pointer

- Pointers hold a memory address
  - Memory address: long
- Value of memory address is a hexadecimal number
- Declare a pointer
  - int *ptr; //pointer
    EX)
    int x = 10;
    ptr=&x; //=200
    cout<<x; //prints 10
    cout<<&x //prints 200
    cout<<&pts; //shows 100 (address for pointer)
    cout<<ptr; //200
    cout<<*ptr; //10
    x=x+5;
    cout<<x //15
    cout<<*ptr //15

  - ptr returns memory address
  - &ptr returns memory adress for ptr
  - *ptr returns value

- Uninitialized pointer
    int *ptr;
    *ptr = 10; //corrupts the value somewhere in the program
    cout<<*ptr;
- Initialized pointer
    int *ptr;
    int x = 20;
    ptr=&x; //initialize pointer
    *ptr=10 //ok
    cout<<x<<endl; //10
    x=x+5;
    cout<<*ptr; //15
- Null pointer
    int *ptr=0 //ptr points to nothing
    int *ptr = null; //ptr points to nothing
    int *ptr = nullptr; //only c++ 11
    *ptr =10 //error
- Reference (variable)
  - Reference is an alias, or an alternative name, to an existing variable
  - Type & refVal = existingVariable
      int x = 10;

```
int &refX = x;
cout<<x; //10
cout << &x; //100
cout<<refX; //10
cout <<&refX; //100
```

- **Call-by-value**

```
int square (int);
int main()
{
	int number=8;
	cout<<"In main: "<<&number<<endl; //200
	cout<<square(number)<<endl; //64
	cout << number<<endl; //8
}
int square(int n)
{
	cout<< "In Square: "<<&n<<endl; //300
	n*=n;
	return n;
}
```

- **Pass by reference with pointer argument**

```
void square(int *)
int main()
{
	int number=8;
	cout<<"In main: "<<&number<<endl; //100
	square (&number);
	cout<<number;//64
	return 0;
}
void Square (int *n)
{
	cout<<"In Square: "<<n<<endl;//8
	*n = *n  *  *n;
	return ;
}
```

- **Pass by reference with reference argument**

```
int square (int &)
int main()
{
	int number = 8;
	cout<<"In Main: "<<&number<<endl; //100
	cout<<square(number)<<endl; //implicitly
	cout<<number<<endl; //64
	return 0;
}
int square (int &n)
```

```
    {
        cout<<"in Square: "<<&n<<endl;
        n *= n;
        return n;
    }
```

Pointers and Arrays

```
// arrnote.cpp
// array accessed with array notation
#include <iostream>
using namespace std;

int main()
   {                                        //array
   int intarray[5] = { 31, 54, 77, 52, 93 };

   for(int j=0; j<5; j++)                   //for each element,
      cout << intarray[j] << endl;          //print value
   return 0;


   }

// array accessed with pointer notation
#include <iostream>
using namespace std;

int main()
   {                                        //array
   int intarray[5] = { 31, 54, 77, 52, 93 };

   for(int j=0; j<5; j++)                   //for each element,
      cout << *(intarray+j) << endl;        //print value
   return 0;


   }

// passarr.cpp
// array passed by pointer
#include <iostream>
using namespace std;
const int MAX = 5;           //number of array elements

int main()
   {
   void centimize(double*);  //prototype

   double varray[MAX] = { 10.0, 43.1, 95.9, 59.7, 87.3 };

   centimize(varray);        //change elements of varray to cm

   for(int j=0; j<MAX; j++)  //display new array values
      cout << "varray[" << j << "]="
           << varray[j] << " centimeters" << endl;
   return 0;
```

```
   }
//-------------------------------------------------------------
void centimize(double* ptrd)
   {
   for(int j=0; j<MAX; j++)
      *ptrd++ *= 2.54;       //ptrd points to elements of varray
```

C-String manipulation

C++ provides following two types of string representations:

- The C-style character string.
- The string class type introduced with Standard C++.

The C-Style Character String:

The C-style character string originated within the C language and continues to be supported within C++. This string is actually a one-dimensional array of characters which is terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

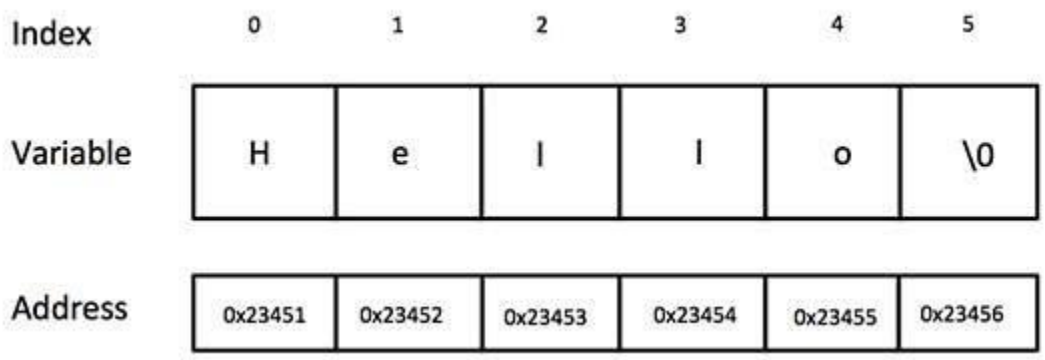
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization, then you can write the above statement as follows:

```
char greeting[] = "Hello";
```

Following is the memory presentation of above defined string in C/C++:



Actually, you do not place the null character at the end of a string constant. The C++ compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print above-mentioned string:

```
#include <iostream>

using namespace std;

int main ()
{
   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

   cout << "Greeting message: ";
   cout << greeting << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
Greeting message: Hello
```

C++ supports a wide range of functions that manipulate null-terminated strings:

	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

Following example makes use of few of the above-mentioned functions:

```
#include <iostream>
#include <cstring>
```

```
using namespace std;

int main ()
{
   char str1[10] = "Hello";
   char str2[10] = "World";
   char str3[10];
   int  len ;

   // copy str1 into str3
   strcpy( str3, str1);
   cout << "strcpy( str3, str1) : " << str3 << endl;

   // concatenates str1 and str2
   strcat( str1, str2);
   cout << "strcat( str1, str2): " << str1 << endl;

   // total lenghth of str1 after concatenation
   len = strlen(str1);
   cout << "strlen(str1) : " << len << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
strcpy( str3, str1) : Hello
strcat( str1, str2): HelloWorld
strlen(str1) : 10
```

C string manipulation

Write a function that returns the number of digits in a given null-terminated string.

```
#include<iostream>
#include<cctype>
using namespace std;
int numAlphas(const char* s)
{
	int count = 0;
	for (int i = 0; s[i] != '\0'; i++)
	{
		if (isdigit(s[i]))
		{
			count++;
		}
	}
	return count;
}

int main()
{
	char str[] = "a12bc3d";
	cout << numAlphas(str);
```

```
}
```

C Strings and Pointers

```
// Create your own strlen function
#include <iostream>
using namespace std;
int myStrLen(char str[]);

int main()
{
	char s[15] = "Hello World";
	cout << myStrLen(s);
	return 0;
}
//------------------------------------------------------------
int myStrLen(char str[])
{
	int i = 0;
	while (str[i] != '\0')
		i++;
	return i;
}
```

Or

```
int myStrLen(char *str)
{
	char *first = str;
	while (*str != '\0')
		str++;
	return str - first;
}
```

Or

```
int myStrLen(char *str)
{
	char *first = str;
	while (*str)
		str++;
	return str - first;

}

// create your own strcpy function
#include <iostream>
```

```
using namespace std;
void myStrcpy(char str2[], char str1[]);

int main()
{
	char s1[15] = "Hello World";
	char s2[30];
	myStrcpy(s2, s1);
	cout << s2;
	return 0;
}
//-------------------------------------------------------------
void myStrcpy(char *to, char * from)
{
	while (*to = *from)
	{
		to++;
		from++;
	}

}
```

Or

```
void myStrcpy(char *to, char * from)
{
	while (*to++ = *from++);

}
```

The String Class in C++:

The standard C++ library provides a **string** class type that supports all the operations mentioned above, additionally much more functionality. We will study this class in C++ Standard Library but for now let us check following example:

At this point, you may not understand this example because so far we have not discussed Classes and Objects. So can have a look and proceed until you have understanding on Object Oriented Concepts.

```
#include <iostream>
#include <string>

using namespace std;

int main ()
{
   string str1 = "Hello";
   string str2 = "World";
   string str3;
   int  len ;
```

```
   // copy str1 into str3
   str3 = str1;
   cout << "str3 : " << str3 << endl;

   // concatenates str1 and str2
   str3 = str1 + str2;
   cout << "str1 + str2 : " << str3 << endl;

   // total lenghth of str3 after concatenation
   len = str3.size();
   cout << "str3.size() :  " << len << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
str3 : Hello
str1 + str2 : HelloWorld
str3.size() :  10
```

Classes

Define a class

```
#include <iostream>
#include <stdio.h>
using namespace std;

class Rectangle {
    int x, y;
  public:
    void set_values (int,int);
    int area (void) {return (x*y);}
};

void Rectangle::set_values (int a, int b) {
  x = a;
  y = b;
}

int main () {
  Rectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
Output
area: 12"
```

Define class with a data member and member functions to set and get its value

```
class MyClass
{
public:
   void setName( string name )
   {
      name = name;
   }

   string getName()
   {
      return name;
   }

   void displayMessage()
   {
      cout << "Welcome to the grade book for  " << getName() << "!" << endl;
   }
private:
   string name;
};

int main()
{
   string n;
   MyClass obj;

   cout << "Initial name is: " << obj.getName() << endl;

   cout << "\nPlease enter the name:" << endl;
   getline( cin, n );
   obj.setName( n );

   cout << endl;
   obj.displayMessage();
   return 0;
}
```

Output

```
Initial name is:

Please enter the name:
Joe

Welcome to the grade book for Joe !
```

Instantiating multiple objects of the class using its constructor

```
class MyClass
{
public:
   MyClass( string name )
   {
```

```
      setName( name );
   }

   void setName( string n )
   {
      name = n;
   }

   string getName()
   {
      return name;
   }

   void displayMessage()
   {
      cout << "Welcome " << getName()  << "!" << endl;
   }
private:
   string name;
};

int main()
{
   MyClass obj1( "A" );
   MyClass obj2( "B" );

   cout << obj1.getName()<< "\n" << obj2.getName() << endl;
   return 0;
}
```

Output
A
B

Class objects can be assigned to each other using default memberwise assignment

```
class Date
{
public:
   Date( int = 1, int = 1, int = 2007 );
   void print();
private:
   int month;
   int day;
   int year;
};

Date::Date( int m, int d, int y )
{
   month = m;
   day = d;
   year = y;
}
void Date::print()
{
```

```
   cout << month << '/' << day << '/' << year;
}

int main()
{
   Date date1( 8, 8, 2008 );
   Date date2;

   cout << "date1 = ";
   date1.print();
   cout << "\ndate2 = ";
   date2.print();

   date2 = date1;

   cout << "\n\nAfter default memberwise assignment, date2 = ";
   date2.print();
   cout << endl;
   return 0;
}
date1 = 8/8/2008
date2 = 1/1/2007

After default memberwise assignment, date2 = 8/8/2008
```