# Review of Strings

# 1 Review of Basic Pointer Concepts

- Point to a memory location.
- Call by reference is based on pointers.
- Operators:
  - `&` *Address of* operator
  - `*` Dereferencing *contents of* operator
- Machine/compiler dependencies exist.
- Care and caution should be exercised when using pointers.

Pointers will be used extensively in later Computer Science courses.

## 1.1 Pointer examples

```
int  a;
int  *aPtr;

a = 5;
cout << a << endl;
aPtr = &a;
cout << *aPtr << endl;    //  contents of a
*aPtr = 6;
cout << a << endl;
cout << *aPtr << endl;    //  contents of a
cout << &a    << endl;    //  address of a
                          //  (compiler/machine dependent)
```

Output:

```
5
5
6
6
0x024b2fa8
```

## 1.2 Arrays and Pointers

```
int  a[5] = { 5, 10, 15, 20, 25 };
int  *aPtr;

aPtr = a;
cout << *aPtr << endl;
aPtr = &a[0];
cout << *aPtr << endl;
aPtr = &a[2];
cout << *aPtr << endl;
```

Output:

```
5
5
15
```

## 1.3 More Arrays and Pointers

Pointer arithmetic.

```
int  a[5] = { 1, 3, 5, 7, 11 };
int  *aPtr;

aPtr = a;
aPtr += 3;      //  advance aPtr by 3
cout << *aPtr << endl;
cout << a[3]  << endl;
```

Output:

```
7
7
```

# 2 Strings

There are numerous functions used for manipulating strings. Most C++ programmers use a *string* class, so these functions are not used/encapsulated within methods of the string class.[1]

Strings may be represented as an array of characters or as a pointer to a character. Some care must be exercised when using pointers.

[1] Not necessarily the standard C++ `string` class.

## Text Processing

> I kinda feel like in teaching developers how complicated Unicode gets we've imbibed the wrong message.
>
> Instead of imbibing "text is hard, we should learn to deal with it", we've imbibed a phobia of everything Unicode.
>
> `@ManishEarth 12:48 AM - 12 Jun 2018`

## 2.1 Character arrays

Character arrays are declared the same as any other array.

Character arrays may be initialized two ways:

```
char str[] = { 't', 'e', 's', 't', '\0' };
```

or

```
char str[] = "test";
```

The compiler takes care of adding the NUL character (`'\0'`) at the end of the string.

## 2.2 Reading strings

```
int main()
{
    const int MAX_FILE_NAME_LENGTH = 32;
    char  inFileName[MAX_FILE_NAME_LENGTH];

    cout << "Input file: ";
    cin >> inFileName;

    cout << "Data will be read from: "
         << inFileName << endl;

    return 0;
}
```

Output:

```
Input file: test.dat
Data will be read from: test.dat
```

### 2.2.1 Alternate methods—using the `get()` method

Replace

```
cin >> inFileName;
```

with

```
cin.get( inFileName, MAX_FILE_NAME_LENGTH );
```

Output:

```
Input file: test2.dat
Data will be read from: test2.dat
```

---

Replace

```
cin >> inFileName;
```

with

```
cin.get( inFileName, MAX_FILE_NAME_LENGTH );
```

Output:

```
Input file: test3.dat test4.dat
Data will be read from: test3.dat test4.dat
```

Note that the results are a bit different than might be expected.

---

Replace

```
cin >> inFileName;
```

with

```
   //  use a space as the "break" point
cin.get( inFileName, MAX_FILE_NAME_LENGTH, ' ' );
```

Output:

```
Input file: test3.dat test4.dat
Data will be read from: test3.dat
```

Note the difference by *tokenizing* on a space.

## 2.3 String Manipulation

### 2.3.1 Example: Converting a string to uppercase

```
#include <iostream>
#include <ctype.h>

using namespace std;

void ConvertStrToUpper( char *s );

main()
{
    char *str1 = "This is a test";
    char str2[] = "Second test";

    cout << "Before converting to upper case:" << endl;
    cout << str1 << endl;
    cout << str2 << endl;

    ConvertStrToUpper( str1 );
    ConvertStrToUpper( str2 );

    cout << "\nAfter converting to upper case:" << endl;
    cout << str1 << endl;
    cout << str2 << endl;
}
```

```
void ConvertStrToUpper( char *s )
{
    while( *s ) {
       if( *s >= 'a' && *s <= 'z' )
           *s = toupper(*s);

       ++s;     //  increment pointer to point
                //  to next character
    }
}
```

Output:

```
Before converting to upper case:
This is a test
Second test

After converting to upper case:
THIS IS A TEST
SECOND TEST
```

## 2.4 Arrays of pointers

```
char *fileNames[3] = {
      "test1.dat",
      "test2.dat",
      "test3.dat"
};

for( int i = 0 ; i < 3 ; ++i )
    cout << fileNames[i] << endl;
```

This is very useful when processing multiple files.

```
for( int i = 0 ; i < 3 ; ++i )
    ProcessFile( fineNames[i] );
```

```
const int N_MONTHS = 12;
char *months[N_MONTHS] = {
      "January", "February", "March",
      "April",   "May",      "June",
      "July",    "August",   "September",
      "October", "November", "December"
};

for( int i = 0 ; i < N_MONTHS ; ++i )
    cout << months[i] << endl;
```

## 2.5 String Length

Finding the length of a string is a very frequently used operation. There are several ways to do it. Most people use the `strlen()` function.

### 2.5.1 Array based method

```
int StringLength( char s[] )
{
    int i = 0;

    while( s[i] != '\0' )
       ++i;

    return i;
}
```

## 2.6 String Comparison

Comparing two strings is another very frequently used operation. There are several ways to do it. Most people use the `strcmp()` function.

Let's write our own string comparison function that behaves the same as `strcmp()`.

- If strings are equal, return 0.
- If the first string is less than string, return -1.
- If the first string is greater than the second, return 1.

### 2.6.1 Array based method

```
int StringCompare( char s1[], char s2[] )
{
    int i;

    for( i = 0 ; s1[i] == s2[i] ; ++i ) {
        if( s1[i] == '\0' )
            return 0;
    }

    return s1[i] - s2[i];
}
```

### 2.6.2 Pointer based method

```
int StringCompare( char *s1, char *s2 )
{
    for( ; *s1 == *s2 ; s1++, s2++ ) {
        if( *s1 == '\0' )
            return 0;
    }

    return *s1 - *s2;
}
```

## 2.7 String-Manipulation Routines

These routines normally operate on NUL-terminated character arrays.

| **Routine** | **Usage** |
|---|---|
| `strcat` | Append one string to another |
| `strchr` | Find first occurrence of specified character in string |
| `strcmp` | Compare two strings |
| `strcpy` | Copy one string to another |
| `strlen` | Find length of string |
| `strncat` | Append n characters to a string |
| `strncmp` | Compare n characters of two strings |
| `strncpy` | Copy n characters of one string to another |
| `strrchr` | Find last occurrence of given character in string |
| `strstr` | Find first occurrence of specified string in another string |
| `strtok` | Find next token in string |