哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

# 高级语言程序设计
# High-level Language Programming

## **Lecture 7** String

Yitian Shao (shaoyitian@hit.edu.cn)
School of Computer Science and Technology

# String

*Course Overview*

- C-strings
- C-string input and output
- C-string functions
- C++ strings
- Character classification

Try it yourself on https://www.onlinegdb.com

# 7.1 C-strings

- C-string： a string is an array of characters (elements of type char) with the null character '\0' in the last element of the array

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

| 'H' | 'e' | 'l' | 'l' | 'o' | '\0' |
|---|---|---|---|---|---|

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

The string literal "Hello“ actually contains 6, rather than 5,characters.

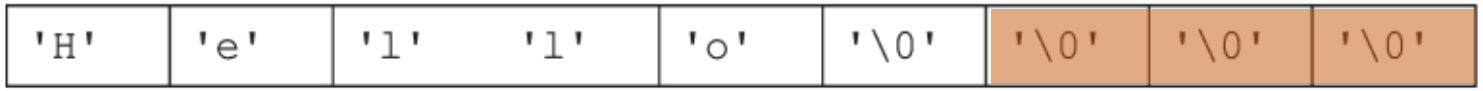
*string literal*

- A sequence of characters enclosed in double quotation marks
- It is not necessary to specify the number of characters
- The compiler automatically inserts the null character '\0' after the last character of a string literal

# 7.1 C-strings

- If specify the size of the array and the string is shorter than this size, the remaining elements of the array are initialized with the **null** character '\0'.

```
char greetings[9] = "Hello" ;
```




- To include a double quote inside a string precede the quote with a back slash (\).

```
char greetings[] = "\"Hello\", I said." ;
cout << greetings ;

"Hello", I said.
```

# 7.1 C-strings

- The newline ('\n') escape sequence can be used in place of *endl* to advance to a new line.

```
cout << "some text\n" ;    ==    cout << "some text" << endl ;
```

- If a string is too long to fit onto a single program line, it can be broken up into smaller segments.

```
char long_string[] = "This is the first half of the string "
                     " and this is the second half." ;
```

# 7.2 C-string input and output

- Remember that the number of elements in the char array must be one more than the number of characters in the string.
- Program Example: a simple demonstration of C-string input and output.

```
4  #include <iostream>
5  using namespace std ;
6
7  main()
8  {
9    const int MAX_CHARACTERS = 10 ;
10   char first_name[ MAX_CHARACTERS + 1 ] ;
11
12   cout << "Enter your first name (maximum "
13        << MAX_CHARACTERS << " characters) " ;
14   cin >> first_name ;
15   cout << "Hello " << first_name << endl ;
16 }
```

```
Enter your first name (maximum 10 characters)
John
Hello John
Enter your first name (maximum 10 characters)
John Paul
Hello John
```

- The extraction operator >> read the characters up to ( but not including) the space character after John.

# 7.2 C-string input and output

- If the user does types in more than 10 characters, what will happen?
  - The array *first_name* will **overflow**, the excess characters will overwrite other areas of memory, and the program will probably malfunction.

```
4  #include <iostream>
5  using namespace std ;
6
7  main()
8  {
9    const int MAX_CHARACTERS = 10 ;
10   char first_name[ MAX_CHARACTERS + 1 ] ;
11
12   cout << "Enter your first name (maximum "
13        << MAX_CHARACTERS << " characters) " ;
14   cin >> first_name ;
15   cout << "Hello " << first_name << endl ;
16 }
```

# 7.2 C-string input and output

***getline()*** function: To adapt user's input and also to **allow for whitespace characters** in the input.

*getline()* reads characters from *cin* until either the user presses the key '\n' or 10 characters have been read
- **'\n'** is called the **delimiter**, when the delimiter is omitted, it is assumed to be '\n'.
- **'\0'** is **automatically added** to the end of a string

- Program Example

```
4  #include <iostream>
5  using namespace std ;
6
7  main()
8  {
9     const int MAX_CHARACTERS = 10 ;
10    char first_name[ MAX_CHARACTERS + 1 ] ;
11
12    cout << "Enter your first name(maximum "
13         << MAX_CHARACTERS << " characters) " ;
14    cin.getline( first_name, MAX_CHARACTERS + 1, '\n' ) ;
15    cout << "Hello " << first_name << endl ;
16 }
```

*Same number as the dimension of the array.*

# 7.2 C-string input and output

- Program Example

```
6  main()
7  {
8    const int MAX_CHARACTERS = 20 ;
9    char student_name[ MAX_CHARACTERS + 1 ] ;
10   int student_number ;
11
12   cout << "Enter student number: " ;
13   cin >> student_number ;
14   cout << "Enter student first name and surname (maximum "
15        << MAX_CHARACTERS << " characters) " ;
16   cin.getline( student_name, MAX_CHARACTERS + 1 ) ;
17   cout << endl << "Data Entered:" << endl
18        << "Student Number: " << student_number << endl
19        << "Student Name: " << student_name << endl ;
20 }
```

```
Enter student number: 12345
Enter student first name and surname (maximum 20 characters)
Data Entered:
Student Number: 12345
Student Name:
```

This line seems to be skipped and no name is read in. Why?

- Line 13 stops reading into the numeric variable student_number as soon as a non-numeric character is read.
- Enter key is pressed after typing 12345 is left in the input stream, which is then read by getline() on line 16 into student_name..

# 7.2 C-string input and output

- Program Example: How to correctly read data into student_name

```
9     const int MAX_CHARACTERS = 20 ;
10    char student_name[ MAX_CHARACTERS + 1 ] ;
11    int student_number ;
12
13    cout << "Enter student number: " ;
14    cin >> student_number ;
15    cout << "Enter student first name and surname (maximum "
16         << MAX_CHARACTERS << " characters) ";
      char dummy ; cin.get( dummy ) ;
18    cin.getline( student_name, MAX_CHARACTERS + 1 ) ;
19    cout << endl << "Data Entered:" << endl
20         << "Student Number: "<< student_number << endl
21         << "Student Name: "<< student_name << endl ;
22  }
```

Solution 1:

Read the newline character '\n' into a 'dummy' character variable. However, if an extra space is entered after the student number, the space is read into the variable dummy, and '\n' is left in the input stream.

# 7.2 C-string input and output

- Program Example: How to correctly read data into student_name

```
9     const int MAX_CHARACTERS = 20 ;
10    char student_name[ MAX_CHARACTERS + 1 ] ;
11    int student_number ;
12
13    cout << "Enter student number: " ;
14    cin >> student_number ;
15    cout << "Enter student first name and surname (maximum "
16         << MAX_CHARACTERS << " characters) ";
17    cin.ignore( 80,'\n' ) ;
18    cin.getline( student_name, MAX_CHARACTERS + 1 ) ;
19    cout << endl << "Data Entered:" << endl
20         << "Student Number: "<< student_number << endl
21         << "Student Name: "<< student_name << endl ;
22 }
```

Solution 2:

discard or ignore all characters in the input stream up to and including the newline character '\n'.

*cin.ignore* (count, delimiter)

# 7.2 C-string input and output

- Program Example: display each character of the string "Hello" on separate lines

Solution:

- Each character of a C-string can be accessed using an index

```
3  #include <iostream>
4  using namespace std ;
5
6  main()
7  {
8    char greetings[6] = "Hello" ;
9    // Display each character of greetings on a new line.
10   for ( int i = 0 ; i < 5 ; i++ )
11   cout << greetings[i] << endl ;
12 }
```

```
H
e
l
l
o
```

# 7.3 C-string functions

- C++ has inherited a library of C-string functions
  - How to use these functions? `#include <cstring>`
  - **Finding the length of a C-string**

***strlen***(): returns the number of characters in a C-string, excluding the null character '\0'.

```
char name1[]   = "Sharon" ;
char name2[10] = "Mark" ;
int len ;
len = strlen( name1 ) ;
cout << setw( 3 ) << strlen( name1 )
     << setw( 3 ) << strlen( name2 )
     << setw( 3 ) << strlen( "Rob" )
     << setw( 3 ) << len ;
```

# 7.3 C-string functions

- **Copying a C-string**
  - ***strcpy***( destination, source )
    - copies the contents of a C-string source to another C-string destination.

```
char name1[] = "Sharon" ;
char name2[10] = "Mark" ;
// Copy the contents of name1 to name2.
strcpy( name2, name1 ) ;
// Restore the original name.
strcpy( name2, "Mark" ) ;
```

Remark:

- A '\0' must be at the end of the string.
- 'destination' string **is assumed big enough** to hold the string being copied to it.

# 7.3 C-string functions

- **C-string concatenation**
  - ***strcat***( str1, str2 )
    - concatenates a C-string str2 to the end of the C-string str1

```
char str1[15] = "first & " ;
char str2[] = "second" ;
strcat( str1, str2 ) ; // str1 is now "first & second".
```

    - Both str1 and str2 must be null-terminated
    - Enough memory must be allocated to str1 to hold the result of the concatenation

# 7.3 C-string functions

- **Comparing C-strings**
  - ***strcmp***( str1, str2 )
    - compares two null-terminated C-strings str1 and str2.
    - This function returns:
      - a negative value, if str1 < str2,
      - 0 , if str1 == str2,
      - a positive value, if str1 > str2.

# 7.3 C-string functions

- Program Example

```
3  #include <iostream>
4  #include <cstring>
5  using namespace std ;
6
7  main()
8  {
9    char password[7] = "secret" ;
10   char user_input[81] ;
11   cout << "Enter Password: " ;
12   cin >>  user_input ;
13   if ( strcmp( password, user_input ) == 0 )
14     cout << "Correct password. Welcome to the system ..." << endl ;
15   else
16     cout << "Invalid password" << endl ;
17 }
```

# 7.3 C-string functions

- strncat( str1, str2, n )
  - Appends the first n characters of the C-string str2 to the C-string str1.
- strncmp( str1, str2, n )
  - Identical to strcmp( str1, str2 ), except that at most, n characters are compared.
- strncpy( str1, str2, n )
  - Copies the first n characters of str2 into str1.

# ASCII Chart

| Code | Symbol | Code | Symbol | Code | Symbol | Code | Symbol |
|---|---|---|---|---|---|---|---|
| 0 | NUL (null) | 32 | (space) | 64 | @ | 96 | ` |
| 1 | SOH (start of header) | 33 | ! | 65 41h | A | 97 61h | a |
| 2 | STX (start of text) | 34 | ” | 66 42h | B | 98 62h | b |
| 3 | ETX (end of text) | 35 | # | 67 43h | C | 99 63h | c |
| 4 | EOT (end of transmission) | 36 | $ | 68 44h | D | 100 64h | d |
| 5 | ENQ (enquiry) | 37 | % | 69 45h | E | 101 65h | e |
| 6 | ACK (acknowledge) | 38 | & | 70 46h | F | 102 66h | f |
| 7 | BEL (bell) | 39 | ’ | 71 ⋮ | G | 103 ⋮ | g |
| 8 | BS (backspace) | 40 | ( | 72 | H | 104 | h |
| 9 | HT (horizontal tab) | 41 | ) | 73 | I | 105 | i |
| 10 | LF (line feed/new line) | 42 | * | 74 | J | 106 | j |
| 11 | VT (vertical tab) | 43 | + | 75 | K | 107 | k |
| 12 | FF (form feed / new page) | 44 | , | 76 | L | 108 | l |
| 13 | CR (carriage return) | 45 | - | 77 | M | 109 | m |
| 14 | SO (shift out) | 46 | . | 78 | N | 110 | n |
| 15 | SI (shift in) | 47 | / | 79 | O | 111 | o |
| 16 | DLE (data link escape) | 48 30h | 0 | 80 | P | 112 | p |
| 17 | DC1 (data control 1) | 49 31h | 1 | 81 | Q | 113 | q |
| 18 | DC2 (data control 2) | 50 32h | 2 | 82 | R | 114 | r |
| 19 | DC3 (data control 3) | 51 33h | 3 | 83 | S | 115 | s |
| 20 | DC4 (data control 4) | 52 34h | 4 | 84 | T | 116 | t |
| 21 | NAK (negative acknowledge) | 53 ⋮ | 5 | 85 | U | 117 | u |
| 22 | SYN (synchronous idle) | 54 ⋮ | 6 | 86 | V | 118 | v |
| 23 | ETB (end of transmission block) | 55 ⋮ | 7 | 87 | W | 119 | w |
| 24 | CAN (cancel) | 56 ⋮ | 8 | 88 | X | 120 | x |
| 25 | EM (end of medium) | 57 39h | 9 | 89 | Y | 121 | y |
| 26 | SUB (substitute) | 58 | : | 90 | Z | 122 | z |
| 27 | ESC (escape) | 59 | ; | 91 | [ | 123 | { |
| 28 | FS (file separator) | 60 | < | 92 | \ | 124 | \| |
| 29 | GS (group separator) | 61 | = | 93 | ] | 125 | } |
| 30 | RS (record separator) | 62 | > | 94 | ^ | 126 | ~ |
| 31 | US (unit separator) | 63 | ? | 95 | _ | 127 | DEL (delete) |

# 7.3 C-string functions

- **Converting numeric C-strings to numbers**
  - The storage of a numeric C-string: in the **ASCII** representation
    - "123“

| '1' | '2' | '3' | '\0' |
|---|---|---|---|
| 49 | 50 | 51 | 0 |
| 00110001 | 00110010 | 00110011 | 00000000 |

  - The storage of an integer value: in **binary**
    - 123

| 00000000 | 01111011 |
|---|---|

# 7.3 C-string functions

- atoi() : C-string to an integer
- atol() : C-string to a long integer
- atof() : C-string to a double float.

```
#include <cstdlib>

char str[] = "123" ;
int int_number ;

double double_number ;

int_number = atoi( str ) ;
double_number = atof( str ) ;
```

Remark:

- Ignore any leading whitespace characters
- Stop converting when a character that cannot be part of the number is reached.
- atoi() will stop when it reaches a decimal point
- atof() will accept a decimal point, because it can be part of a decimal number.

# 7.3 C-string functions

| Function | Remark |
|---|---|
| strlen(str) | Finding the length of a C-string str |
| strcpy(str1, str2) | copies the contents of a C-string str2 to str1 |
| strcat( str1, str2 ) | concatenates a C-string str2 to the end of the C-string str1 |
| strcmp( str1, str2 ) | compares two null-terminated C-strings str1 and str2. |
| strncat( str1, str2, n ) | Appends the first n characters of the C-string str2 to str1. |
| strncmp( str1, str2, n ) | Identical to strcmp, except that at most n characters are compared. |
| strncpy( str1, str2, n ) | Copies the first n characters of str2 into str1. |

# 7.4 C++ strings

Common errors in C-strings:

- Attempting to access elements **outside the array bounds**
- Not using the function **strcpy**() to copy C-strings
- Not using **strcmp**() to compare two C-strings.
- …

C++ strings

- The string data type is **not** built into C++
- In C++, the string data type is **defined by a class**

# 7.4 C++ strings

- Program Example

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string password = "secret" ;
    string user_input ;
    cout << "Enter Password: " ;
    cin >> user_input ;
    if ( password == user_input )
        cout << "Correct password. Welcome to the system ..." << endl ;
    else
        cout << "Invalid password" << endl ;
    return 0;

}
```

Remark:

- C++ strings are compared using **==, <, >**,etc instead of *strcmp*() in C-strings.
- C++ has many useful string member functions.

# 7.4 C++ strings

- **string initialization and assignment**

```
// String initialisation examples.
string str1 = "ABCDEFGHI" ; // Define a string and initialise it.
string str2( 11, '-' ) ;    // Define a string of 11 dashes.
string str3 = "This is the first part"
              " and this is the second part." ;
string str4 = str2 ;  // Initialise str4 with str2.
string str5 ; // str5 has no initial value.

cout << "After initialisations:" << endl
     << "  str1=" << str1 << endl
     << "  str2=" << str2 << endl
     << "  str3=" << str3 << endl
     << "  str4=" << str4 << endl
     << "  str5=" << str5 << endl ;
```

```
After initialisations:
  str1=ABCDEFGHI
  str2=-----------
  str3=This is the first part and this is the second part.
  str4=-----------
  str5=
```

- str2 shows how to assign a C++ string with a number of identical characters
- The string that is being assigned can be on two or more lines.
- str5 is called an empty string.

# 7.4 C++ strings

- **string initialization and assignment**

```
24   // String assignment examples.
25   str1 = "ABCD" ;
26   str2.assign( 3, '.' ) ; // Assign 3 dots to str2.
27   cout << "After the 1st and 2nd assignments:" << endl
28        << "  str1=" << str1 << endl
29        << "  str2=" << str2 << endl ;
33   str5.assign( str1, 1, 3 ) ;  // Assign "BCD" to str5.
34   cout << "After the 3rd assignment:" << endl
35        << "  str5=" << str5 << endl ;
```

```
After the 1st and 2nd assignments:
  str1=ABCD
  str2=...
After the 3rd assignment:
  str5=BCD
```

line33 assign(): assign part of str1 to str5.

- *str1* is the string to assign from
- *1* is the **starting position**
- *3* is the **number of characters to assign**

# 7.4 C++ strings

- **string swap**

```
37    // Swapping strings.
38    cout << "Before swapping str1 and str2:"  << endl
39         << "  str1=" << str1 << endl
40         << "  str2=" << str2 << endl ;
```

```
Before swapping str1 and str2:
  str1=ABCD
  str2=...
After swapping str1 and str2:
  str1=...
  str2=ABCD
```

```
41    str1.swap( str2 ) ; // swap str1 and str2.
42    cout << "After swapping str1 and str2:"  << endl
43         << "  str1=" << str1 << endl
44         << "  str2=" << str2 << endl ;
45 }
```

# 7.4 C++ strings

- **string concatenation**

```
string str1 = "ABCD", str2, str3 ;

str2.assign( 3, '.' ) ; // Assign 3 dots to str2.

str3 = str1 + str2 ;  // With strings, + means concatenate.
```

```
After the 1st concatenation:
   str1=ABCD
   str2=...
   str3=ABCD...
```

# 7.4 C++ strings

- **string concatenation**

```
string str1 = "ABCD", str2, str3 ;

str2.assign( 3, '.' ) ; // Assign 3 dots to str2.

str3 = str1 + str2 ;  // With strings, + means concatenate.

str3 += "etc." ;    // same as str3 = str3 +

str3.append ( ", etc., etc." ) ;

str3.append( str4, 11, 7 ) ;  // Append "the end" to str3.

str3.append( 3, '.' ) ;  // Append 3 dots.
```

```
After the 1st concatenation:
  str1=ABCD
  str2=...
  str3=ABCD...
```

```
After the 2nd concatenation:
  str3=ABCD...etc.
```

```
After the 3rd concatenation:
  str3=ABCD...etc., etc., etc.
```

```
After the 4th concatenation:
  str3=This is the end
```

```
After the 5th concatenation:
  str3=This is the end...
```

# 7.4 C++ strings

- **string length, string indexing and sub-strings**

```
string str1 = "ABCDEFGH" ;
int len1 ;

// Store the length of str1 in len1.
len1 = str1.length() ;

str1[0] = '*' ;
str1[len1-1] = '*' ;
str1.at( 0 ) = 'A' ;
str1.at( len1 - 1 ) = 'H' ;

// Display a space between each character of str1.
cout << str1 << " with a space between each character: " << endl ;
for ( int i = 0 ; i < len1 ; i++ )
    cout << str1.at( i ) << ' ' ;
```

- Change the first and last characters
- Check the index value to ensure it is not out of range by ***at()***.

```
ABCDEFGH with a space between each character:
A B C D E F G H
```

# 7.4 C++ strings

```
string str2 = "ABCDEFGH" ;
cout << "Demonstration of substr:" << endl << "   " ;
cout << "The first four characters of " << str2<< " are "
     << str2.substr( 0, 4 ) << endl << "   "
     << "The middle two characters of " << str2 << " are "
     << str2.substr( 3, 2 ) << endl << "   "
     << "The last three characters of " << str2 << " are "
     << str2.substr( 5,3 ) << endl ;
```

```
Demonstration of substr:
  The first four characters of ABCDEFGH are ABCD
  The middle two characters of ABCDEFGH are DE
  The last three characters of ABCDEFGH are FGH
```

- ***substr***() can extract part of a C++ string.
- The 1st argument is a starting position
- 2nd argument is the number of characters to extract.

# 7.4 C++ strings

- **string replace, erase, insert and empty strings**

```
string str1 =  " ABCDE "  ;
string str2 =  " abcdefghij "  ;

str1.replace( 1, 3, str2, 2, 4 ) ;
cout <<  " After the 1st replacement: "  << endl
     <<  "  str1= "  << str1 << endl ;

str1 =  " ABCDE "  ;
str1.replace( 1, 3, str2 ) ;
cout <<  " After the 2nd replacement: "  << endl
     <<  "  str1= "  << str1 << endl ;

str1.erase( 9 ) ;
cout <<  " After the 1st erase: "  << endl
     <<  "  str1= "  << str1 << endl ;
```

- Character at position 0 is the first character.
- Replace 3 characters from str1 starting at the 2nd character position with 4 characters from str2, starting at the 3rd character position.

```
After the 1st replacement:
  str1=AcdefE
```

# 7.4 C++ strings

```
string str1 =  " ABCDE "  ;
string str2 =  " abcdefghij "  ;

str1.replace( 1, 3, str2, 2, 4 ) ;
cout <<  " After the 1st replacement: "  << endl
     <<  "  str1= "  << str1 << endl ;

str1 =  " ABCDE "  ;
str1.replace( 1, 3, str2 ) ;
cout <<  " After the 2nd replacement: "  << endl
     <<  "  str1= "  << str1 << endl ;

str1.erase( 9 ) ;
cout <<  " After the 1st erase: "  << endl
     <<  "  str1= "  << str1 << endl ;
```

```
After the 2nd replacement:
  str1=AabcdefghijE
```

- Erase from the 10th character position to the end of str1.

```
After the 1st erase:
  str1=Aabcdefgh
```

# 7.4 C++ strings

```
// str1 = Aabcdefgh
str1.erase( 4, 2 ) ;
cout <<  " After the 2nd erase: "  << endl
     <<  "  str1= "  << str1 << endl ;

str1.erase() ;
cout <<  " After the 3rd erase: "  << endl
     <<  "  str1= "  << str1 << endl ;
if ( str1.empty() )
    cout <<  " str1 is empty "  << endl ;
else
    cout <<  " str1 is not empty "  << endl ;

str1 =  " ABCDEFG "  ;
str1.insert( 4, str2, 1, 6 ) ;
cout <<  " After the 1st insert: "  << endl
     <<  "  str1= "  << str1 << endl ;
```

- Erase 2 characters starting at the 5th character position.

- Erase the entire string.

- Judge whether str1 is empty?

- Starting at the 2nd character of str2, insert 6 characters at the 5th character position of str1.

```
After the 2nd erase:
  str1=Aabcfgh

After the 3rd erase:
  str1=

  str1 is empty

After the 1st insert:
  str1=ABCDbcdefgEFG
```

# 7.4 C++ strings

- **string searching**

```
string str1 = " ABCDEFABCDEF " ;
int p ;

p = str1.find( " CDE " ) ;
if ( p == -1 )
    cout << " CDE Not Found in str1 " << endl ;
else
    cout << " First Occurrence of CDE Found at "
         << p << endl ;

p = str1.rfind( " CDE " ) ;
if ( p == -1 )
    cout << " CDE Not Found " << endl ;
else
    cout << " Last Occurrence of CDE Found at "
         << p << endl ;

p = str1.find_first_of( " ED " ) ;
if ( p == -1 )
    cout << " E or D Not Found in str1 " << endl ;
else
    cout << " E or D First Found at " << p << endl ;
```

• Hold the position of the first occurrence of "CDE" in str1. If "CDE" is not in str1, p = -1.

```
First Occurrence of CDE Found at 2
```

•Reverse find the last occurrence of "CDE".

```
Last Occurrence of CDE Found at 8
```

•Find the first occurrence of any one of a number of characters.

```
E or D First Found at 3
```

# 7.4 C++ strings

- **string searching**

```
string str1 = " ABCDEFABCDEF " ;
int p ;

p = str1.find_last_of( " ED " ) ;
if ( p == -1 )
    cout << " E or D Not Found in str1 " << endl ;
else
    cout << " E or D Last Found at " << p << endl ;

p = str1.find_first_not_of( " ABC " ) ;
if ( p == -1 )
    cout << " No Characters Other than A, B or C Found in str1 "
         << endl ;
else
    cout << " A Character Other than A, B or C First Found at "
         << p << endl ;

p = str1.find_last_not_of( " ABC " ) ;
if ( p == -1 )
    cout << " No Characters Other Than A, B or C Found in str1 "
         << endl ;
else
    cout << " A Character Other Than A, B or C Last Found at "
         << p << endl ;
```

- Find the last occurrence of any one of a number of characters.

```
E or D Last Found at 10
```

- Find the first occurrence of any character that is not one of a number of characters.

```
A Character Other than A, B or C First Found at 3
```

- Find the last occurrence of any character that is not one of a number of characters.

```
A Character Other Than A, B or C Last Found at 11
```

# 7.4 C++ strings

- **string comparisons**

```
string str1 =  " ABCDEFGH "  ;
string str2 =  " BCD "  ;
int result ;

if ( str1 == str2 )
    cout <<  " str1 and str2 are equal "  << endl ;
if ( str1 < str2 )
    cout <<  " str1 is less than str2 "  << endl ;
if ( str1 > str2 )
    cout <<  " str1 is greater than str2 "  << endl ;

result = str1.compare( str2 ) ;
if ( result == 0 )
    cout <<  " str1 and str2 are equal "  << endl ;
if ( result < 0 )
    cout <<  " str1 is less than str2 "  << endl ;
if ( result > 0 )
    cout <<  " str1 is greater than str2 "  << endl ;
```

- C++ strings can be compared with the standard comparison operators : ==, !=, <=, >= < and >

- result is < 0 if the first differing character in str1 is less than the character in the same position in str2.
- result is 0 if all the characters of str1 and str2 are equal and the two strings are the same length.
- Otherwise result is > 0.

```
str1 is less than str2
```

# 7.4 C++ strings

- **string comparisons**

```
string str1 = "ABCDEFGH" ;
string str2 = "BCD" ;
int result ;

result = str1.compare( 1, 3, str2 ) ;
if ( result == 0 )
    cout << "Characters 2,3 and 4 of str1" << endl <<
         " and all the characters of str2 are equal" << endl ;
if ( result < 0 )
    cout << "Characters 2,3 and 4 of str1 are less than"
    << endl << " all the characters of str2" << endl ;
if ( result > 0 )
    cout << "Characters 2,3 and 4 of str1 are greater than"
         << endl << " all the characters of str2" << endl ;
```

- Compares a 3-character substring of str1 starting at the second character with all the characters of str2.

```
Characters 2,3 and 4 of str1
and all the characters of str2 are equal
```

# 7.4 C++ strings

- Just like any other data type, **arrays of C-strings** and **arrays of C++ strings** can be defined.
- Arrays of C++ strings are much easier to work with.

```
main()
{
  // Define an array of twelve strings.
  string months[12] = { "January", "February", "March",
                        "April", "May", "June", "July",
                        "August", "September", "October",
                        "November", "December" } ;

  // Display the months of the year.
  cout <<  "The months of the year are:" << endl ;
  for ( int i = 0 ; i < 12 ; i++ )
    cout << months[i] << endl ;
}
```

# 7.5 Character classification

- There are a number of C++ functions that can be used to test the value of a single character. These functions return a true (non-zero integer) value or a false (zero integer) value depending on whether or not the character belongs to a particular set of characters.

| Function | Character set |
|---|---|
| `isalnum` | Alphanumeric character: A-Z, a-z, 0-9 |
| `isalpha` | Alphabetic character: A-Z, a-z |
| `isascii` | ASCII character: ASCII codes 0-127 |
| `iscntrl` | Control character: ASCII codes 0-31 or 127 |
| `isdigit` | Decimal digit: 0-9 |
| `isgraph` | Any printable character other than a space |
| `islower` | Lowercase letter: a-z |
| `isprint` | Any printable character, including a space |
| `ispunct` | Any punctuation character |
| `isspace` | Whitespace character: \t,\v,\f,\r,\n or space ASCII codes 9-13 or 32 |
| `isupper` | Uppercase letter: A-Z |
| `isxdigit` | Hexadecimal digit: 0-9 and A-F |

# 7.5 Character classification

- C++ has also two further functions that are used to covert the case of a character: tolower and toupper.

| Function | Purpose |
|---|---|
| `tolower` | Converts an uppercase character to lowercase. |
| `toupper` | Converts a lowercase character to uppercase. |

# 7.5 Character classification

```
string in_name ;

cout << "Type a forename and surname and press Enter: " ;
getline( cin, in_name ) ;

// Ignore all characters until the first alphabetical character
// of the forename is reached.
int i=0 ;
while( !isalpha( in_name[i] ) )
  i++ ;

// Capitalise the first character of the forename.
in_name[i] = toupper( in_name[i] ) ;

// Ignore all characters in forename until a space is reached.
while( !isspace( in_name[i] ) )
  i++ ;

// Ignore all characters until the first letter of the surname
// is reached.
while( !isalpha( in_name[i] ) )
  i++ ;

// Capitalise the first letter of the surname.
in_name[i] = toupper( in_name[i] ) ;

cout << "Formatted Name: " << in_name << endl ;
```

A sample run of this program:

```
Type a forename and surname and press Enter: john smith
Formatted Name: John Smith
```

# In-class quiz

On your paper, write down your **name + student ID**, and **C++ code** that does the following:

1. Include the **iostream** and C++ **string** library
2. Create a **string** variable, and allow the user to input a **string value** via the **cin** command
3. Check if the first character **(char)** of the **string value** is a alphabetic character or not, print the result to the terminal using **cout** command
4. Use a "for" loop to check all characters of the entire string

```
string str1 = "ABCDEFGH" ;
int len1 ;

// Store the length of str1 in len1.
len1 = str1.length() ;

str1[0] = '*' ;
str1[len1-1] = '*' ;
```

get string length

string indexing

| Function | Character set |
|---|---|
| isalnum | Alphanumeric character: A-Z, a-z, 0-9 |
| isalpha | Alphabetic character: A-Z, a-z |
| isascii | ASCII character: ASCII codes 0-127 |
| iscntrl | Control character: ASCII codes 0-31 or 127 |
| isdigit | Decimal digit: 0-9 |
| isgraph | Any printable character other than a space |
| islower | Lowercase letter: a-z |
| isprint | Any printable character, including a space |
| ispunct | Any punctuation character |
| isspace | Whitespace character: \t,\v,\f,\r,\n or space ASCII codes 9-13 or 32 |
| isupper | Uppercase letter: A-Z |
| isxdigit | Hexadecimal digit: 0-9 and A-F |

# HOMEWORK

# Homework 7

- 1. What is the output from the following program segment?

```
char str1[] = " abc " ;
char str2[] = " ABCD " ;
cout << str1 << endl << strlen( str1 ) << endl ;
if ( strcmp( str1, str2 ) == 0 )
    cout << str1 << " == " << str2 << endl ;
else
    if ( strcmp( str1, str2 ) < 0 )
        cout << str1 << " < " << str2 << endl ;
else
    if ( strcmp( str1, str2 ) > 0 )
        cout << str1 << " > " << str2 << endl ;
char str3[8] ;
strcpy( str3, str1 ) ;
strcat ( str3, str2 ) ;
cout << str3 << endl << strlen( str3 ) << endl ;
str3[6] = 'x' ;
cout << str3 << endl ;
```

# Homework 7

- 2. Modify exercise 1 to use C++ strings rather than C-strings.
- 3. Given the following:

  char c_str1[18] ;

  char c_str2[6] = " abcde " ;

  what is in c_str1 after each of the following?

  (a ) strcpy( c_str1, " A string " ) ;

  (b) strcat( c_str1, " of text. " ) ;

  (c) strncpy( c_str1, c_str2, 1 ) ;

# Homework 7

- 4. Write a program to input a C-string from the keyboard and replace each space in the string with the character '_'.
- 5. What is the output from the following program segment?

```
string str = "ABCDEFGHIJ" ;
cout << str << endl << str.length() << endl ;
str.replace( 4, 2, "123456" ) ;
str.at( 3 ) = '0' ;
cout << str << endl << str.length() << endl ;
str.erase( 10, 2 ) ;
cout << str << endl << str.length() << endl ;
cout << str.substr( 3, 7 ) << endl ;
str += "KLMN" ;
cout << str << endl << str.length() << endl ;
str.insert( 10, "7890" ) ;
cout << str << endl << str.length() << endl ;
cout << str.find( "0" ) << endl ;
```

# Homework 7

- 6. Write a program to input a string. If every character in the string is a digit ('0' to '9'), then convert the string to an integer, add 1 to it, and display the result. If any one of the characters in the string is not a digit, display an error message.