**Computer programming (STQS1313)**
**Sem 2, 2020/2021**
**Notes - String class and C-strings**

There are two ways of creating a string:

- string class; example:

```
string test = “abcd”;
```

- character string (C-string): array of characters terminated by a special end-of-string marker called the NULL character; example:

```
char test[5] = {‘a’, ‘b’, ‘c’, ‘d’, ‘\0’}; or
char test[5] = “abcd”;
```

This is some introduction about these two topics taken from the book by Bronson:

*Manipulating strings stored as* ***C-strings*** *can be time consuming. This is especially true for applications requiring numerous string operations, such as inserting, searching, and/or deleting characters in an existing string.*

*The reason is that each time a string is lengthened by adding characters, a new and larger array must be created, and removing characters requires shifting characters to fill the empty spaces left by deleted characters, with adjustment of the end-of-string null character. Searching for specific characters in a string requires nested loops.*

*To circumvent the coding required for these types of operations, C++ provides the* ***string class*** *as part of the standard C++ library. This class provides an expanded set of class functions, including easy insertion and removal of characters from a string, automatic string expansion when a string’s original capacity is exceeded, string contraction when characters are removed from a string, and range checking to detect invalid character positions. In many ways, the strings created from the string class can also be manipulated by using the array techniques suitable for C-strings. The* ***main difference*** *is that string class strings aren’t terminated with a null character, and the string class provides many useful functions for operating on strings.*

***Exception handling*** *is a means of error detection and processing that has gained increasing acceptance in programming technology. It permits detecting an error at the point in the code where the error has occurred and provides a means of processing the error and returning control to the line that generated the error. As you also see, although error detection and code correction are possible by using if statements and functions, exception handling is one more useful programming tool for validating user input.*

We are going to learn a few things in this topic, which are:

- string class
  - ➢ background
  - ➢ string class functions
  - ➢ string input and output
  - ➢ string processing
  - ➢ character manipulation methods
- C-strings
  - ➢ background
  - ➢ C-String Input and Output
  - ➢ C-string processing
  - ➢ C-string manipulation functions
  - ➢ character-handling functions

## I) STRING CLASS

### Background

1) Requires a standard library: the *string* class. So, we need to have

```
        #include<string>
```

2) Storage area of data is called an **object** rather than a variable.
3) From the line

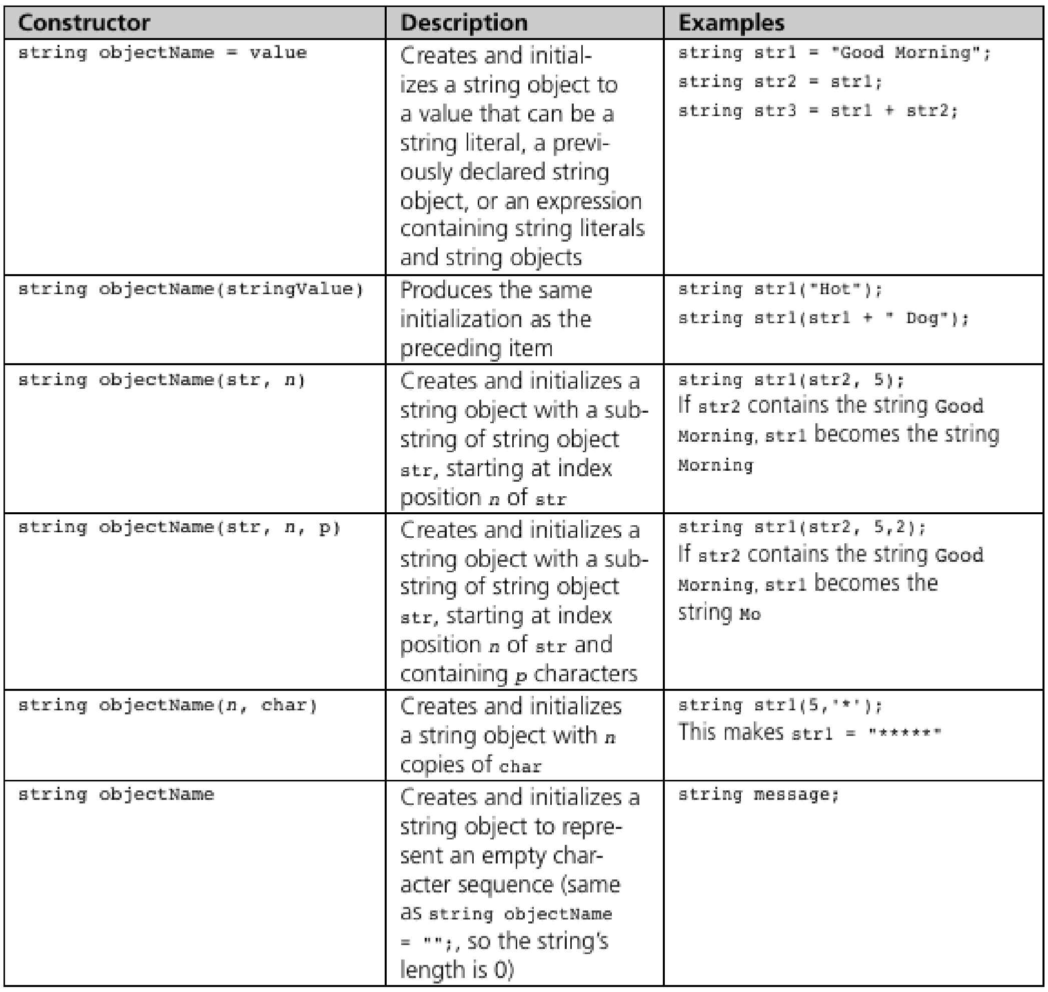
```
            string test = "abcd";
```

   the `abcd` is called a string literal, string value, string constant, or simply a string.

4) The process of creating a string object, as shown above, is also called **instantiating** a string object.
5) The quotation marks indicate the beginning and ending points of the string and are never stored with the sting.
6) The first character in a string is always designated as position 0.

### String class functions

1) In a class, functions are formally referred to as **methods**.
2) the methods that perform the tasks of creating and initialising are called **constructor methods**, or just **constructors**.
3) examples of constructor (from Bronson's book):

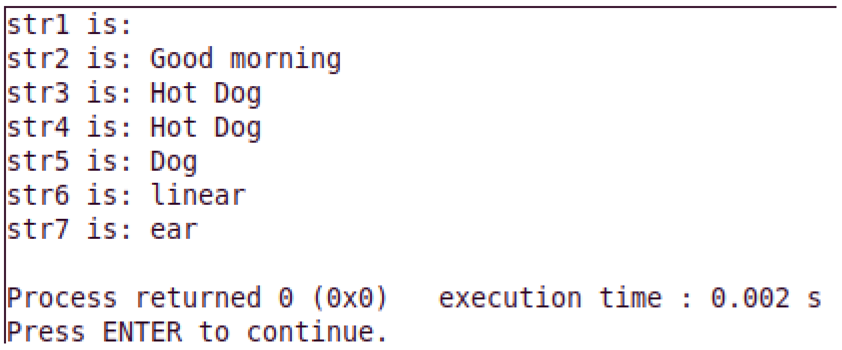


| Constructor | Description | Examples |
|---|---|---|
| `string objectName = value` | Creates and initializes a string object to a value that can be a string literal, a previously declared string object, or an expression containing string literals and string objects | `string str1 = "Good Morning";`<br>`string str2 = str1;`<br>`string str3 = str1 + str2;` |
| `string objectName(stringValue)` | Produces the same initialization as the preceding item | `string str1("Hot");`<br>`string str1(str1 + " Dog");` |
| `string objectName(str, n)` | Creates and initializes a string object with a substring of string object `str`, starting at index position *n* of `str` | `string str1(str2, 5);`<br>If `str2` contains the string `Good Morning`, `str1` becomes the string `Morning` |
| `string objectName(str, n, p)` | Creates and initializes a string object with a substring of string object `str`, starting at index position *n* of `str` and containing *p* characters | `string str1(str2, 5,2);`<br>If `str2` contains the string `Good Morning`, `str1` becomes the string `Mo` |
| `string objectName(n, char)` | Creates and initializes a string object with *n* copies of `char` | `string str1(5,'*');`<br>This makes `str1 = "*****"` |
| `string objectName` | Creates and initializes a string object to represent an empty character sequence (same as `string objectName = "";`, so the string's length is 0) | `string message;` |

- Code 64

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
     // Seven ways to instantiate (create) a string object
     string str1;                      // create an empty string named str1
     string str2("Good morning");
     string str3 = "Hot Dog";
     string str4(str3);
     string str5(str4, 4);
     string str6 = "linear";
     string str7(str6, 3, 3);

     cout << "str1 is: " << str1 << endl;
     cout << "str2 is: " << str2 << endl;
     cout << "str3 is: " << str3 << endl;
     cout << "str4 is: " << str4 << endl;
     cout << "str5 is: " << str5 << endl;
     cout << "str6 is: " << str6 << endl;
     cout << "str7 is: " << str7 << endl;

     return 0;
}
```

with output as follows:



```
str1 is:
str2 is: Good morning
str3 is: Hot Dog
str4 is: Hot Dog
str5 is: Dog
str6 is: linear
str7 is: ear

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

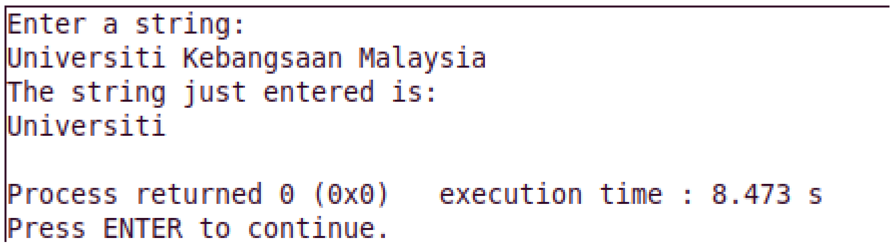
1) In Code 64:
   - ➢ string objects are created in a similar manner as declaring variables, but instead of using built-in data types, such as `int` or `double`, the keyword `string` is used.
   - ➢ each string, except the first one (`str1`), has been initialised explicitly.
   - ➢ no explicit string is used to initialize `str1`, so it's initialized automatically with no characters at all. These strings are referred to as "empty strings".
   - ➢ Because the first character in a string is designated as position 0, not 1, the character position of `D` in the string `Hot Dog`, for example, is position 4, as shown below:

## String Input and Output

1) 3 main methods
   - `cout`
   - `cin`: input method that stops reading string input when white space (any combination of blank spaces, tabs, or new lines) is encountered.
   - `getline(cin,strObj)`: input method that inputs all characters entered, stores them in `strObj` , and stops accepting characters when it receives a newline character ('`\n`') i. e. when we hit Enter.
2) because a blank space and other **white space characters** terminate a `cin` extraction operation, `cin`'s usefulness for entering string data is restricted; therefore, `getline()` is used when we have a sentence.
3) Code 65:

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
   string message;   // declaring a string object

   cout << "Enter a string:\n";
   cin  >> message;

   cout << "The string just entered is: \n"
        << message << endl;

   return 0;
}
```

gives output



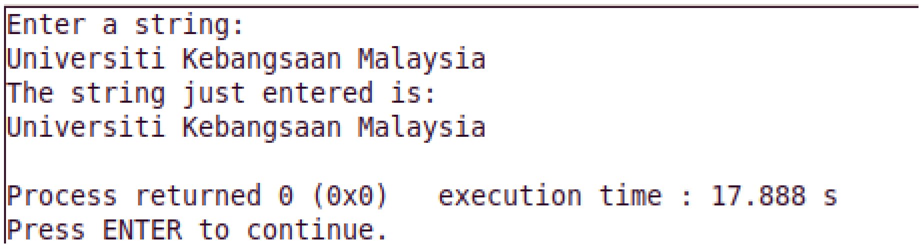
```
Enter a string:
Universiti Kebangsaan Malaysia
The string just entered is:
Universiti

Process returned 0 (0x0)   execution time : 8.473 s
Press ENTER to continue.
```

when given input “Universiti Kebangsaan Malaysia”.

4) Code 66:

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
   string message;   // declaring a string object

   cout << "Enter a string:\n";
   getline(cin,message);

   cout << "The string just entered is: \n"
        << message << endl;
   return 0;
}
```

gives a correct output



```
Enter a string:
Universiti Kebangsaan Malaysia
The string just entered is:
Universiti Kebangsaan Malaysia

Process returned 0 (0x0)   execution time : 17.888 s
Press ENTER to continue.
```

when given input “`Universiti Kebangsaan Malaysia`”.

5) In its most general form, the `getline()` method has the following syntax:

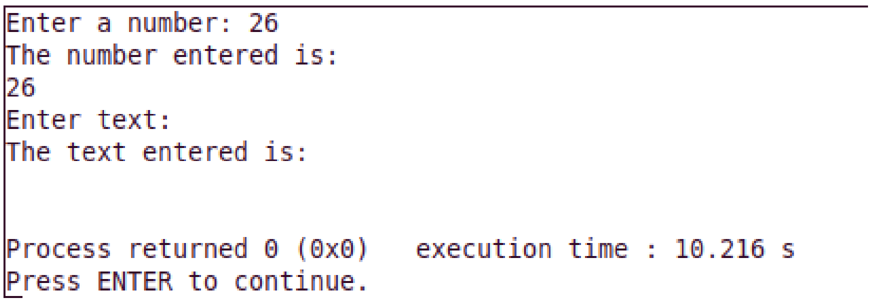
```
getline(cin,strObj,terminatingChar)
```

- here, the third argument, `terminatingChar`, is an optional character constant, or variable, specifying the terminating character.
- if it is omitted when `getline()` is called (as in Code 66), the default terminating character is the newline (‘`\n`’) character, i. e. when we hit Enter.

6) The phantom newline character.
- seemingly strange results can happen when the `cin` and `getline()` method are used together to accept data or when `cin` is used by itself to accept characters.
- Code 67:

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
    int value;
    string message;

    cout << "Enter a number: ";
    cin  >> value;
    cout << "The number entered is:\n"
         << value << endl;

    cout << "Enter text:\n";
    getline(cin,message);
    cout << "The text entered is: \n"
         << message << endl;
    return 0;
}
```

- Below is the output that appears immediately when 26 is entered:

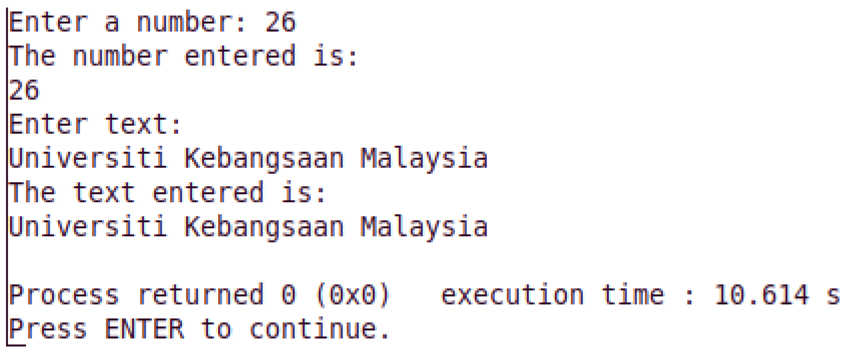


```
Enter a number: 26
The number entered is:
26
Enter text:
The text entered is:


Process returned 0 (0x0)   execution time : 10.216 s
Press ENTER to continue.
```

- Notice that in entering a number as prompted, we type the number 26 and press the Enter key.
- The next input statement, which is a call to `getline()`, picks up the code for the Enter key as the next character and terminates any further input.
- Therefore, no text is accepted in response to the prompt `Enter text:`
- One of the ways to solve this issue is by placing `cin.ignore()` after the `cin` (Code 68):

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
    int value;
    string message;

    cout << "Enter a number: ";
    cin  >> value;
    cin.ignore();
    cout << "The number entered is:\n"
         << value << endl;

    cout << "Enter text:\n";
    getline(cin,message);
    cout << "The text entered is: \n"
         << message << endl;
    return 0;
}
```

which gives the output that we expected:

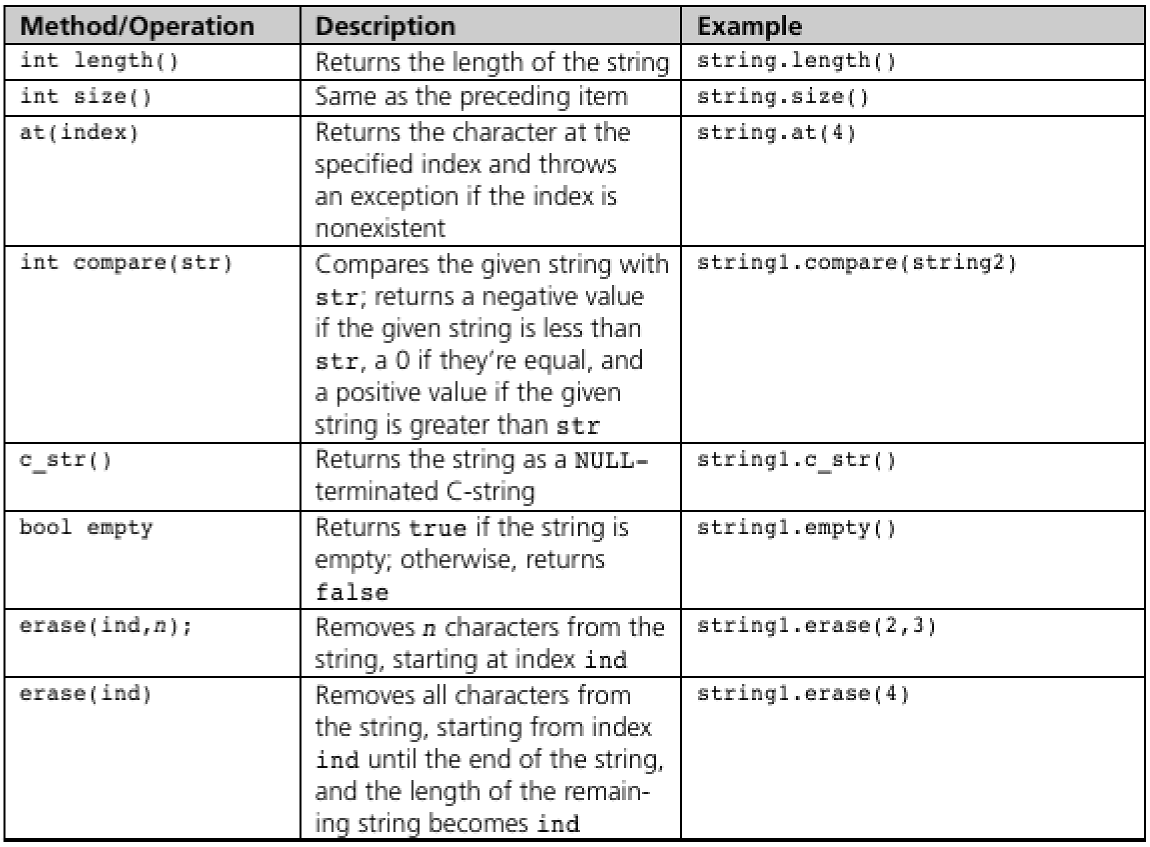


```
Enter a number: 26
The number entered is:
26
Enter text:
Universiti Kebangsaan Malaysia
The text entered is:
Universiti Kebangsaan Malaysia

Process returned 0 (0x0)   execution time : 10.614 s
Press ENTER to continue.
```

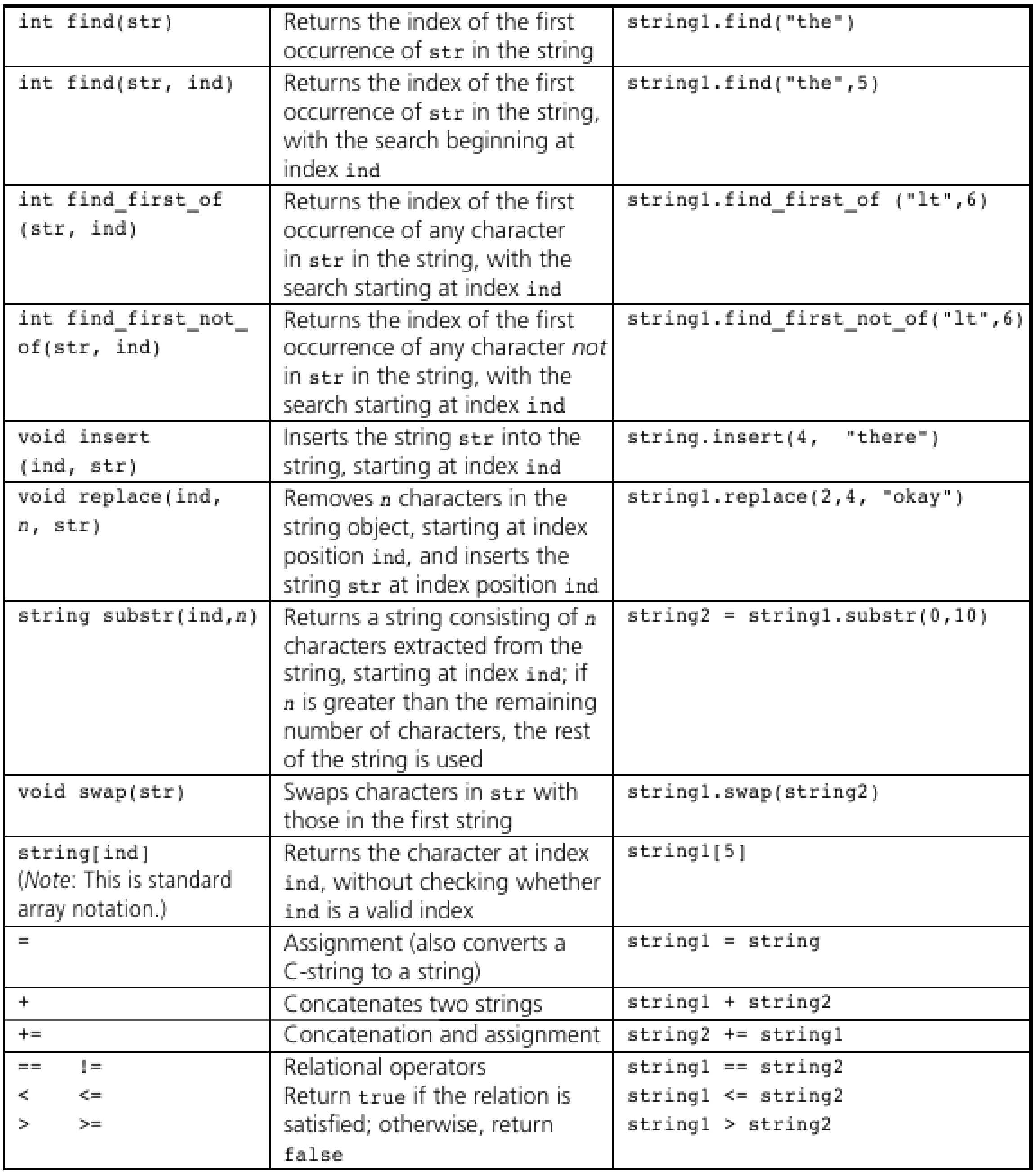
- Another solution, which is preferred, is to avoid mixing `cin` with `getline()` inputs in the same program.

**String processing**

1) Strings can be manipulated by using:
   - **string class methods**, or
   - **character-at-a-time methods** (character manipulation methods).
2) The most commonly used string classs method are shown below:



| Method/Operation | Description | Example |
|---|---|---|
| `int length()` | Returns the length of the string | `string.length()` |
| `int size()` | Same as the preceding item | `string.size()` |
| `at(index)` | Returns the character at the specified index and throws an exception if the index is nonexistent | `string.at(4)` |
| `int compare(str)` | Compares the given string with `str`; returns a negative value if the given string is less than `str`, a 0 if they're equal, and a positive value if the given string is greater than `str` | `string1.compare(string2)` |
| `c_str()` | Returns the string as a `NULL`-terminated C-string | `string1.c_str()` |
| `bool empty` | Returns `true` if the string is empty; otherwise, returns `false` | `string1.empty()` |
| `erase(ind,n);` | Removes *n* characters from the string, starting at index `ind` | `string1.erase(2,3)` |
| `erase(ind)` | Removes all characters from the string, starting from index `ind` until the end of the string, and the length of the remaining string becomes `ind` | `string1.erase(4)` |

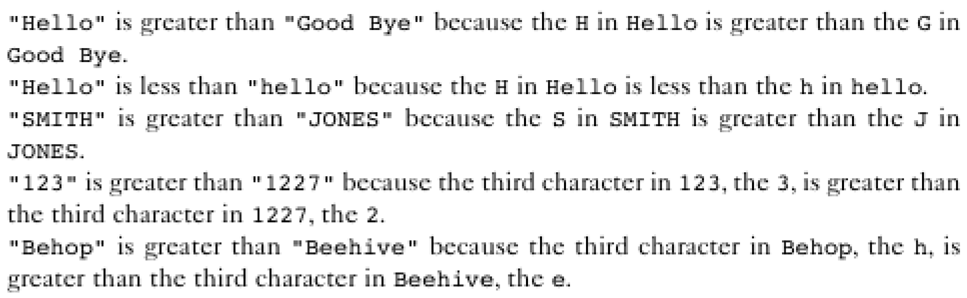


| | | |
|---|---|---|
| `int find(str)` | Returns the index of the first occurrence of `str` in the string | `string1.find("the")` |
| `int find(str, ind)` | Returns the index of the first occurrence of `str` in the string, with the search beginning at index `ind` | `string1.find("the",5)` |
| `int find_first_of (str, ind)` | Returns the index of the first occurrence of any character in `str` in the string, with the search starting at index `ind` | `string1.find_first_of ("lt",6)` |
| `int find_first_not_ of(str, ind)` | Returns the index of the first occurrence of any character *not* in `str` in the string, with the search starting at index `ind` | `string1.find_first_not_of("lt",6)` |
| `void insert (ind, str)` | Inserts the string `str` into the string, starting at index `ind` | `string.insert(4, "there")` |
| `void replace(ind, n, str)` | Removes *n* characters in the string object, starting at index position `ind`, and inserts the string `str` at index position `ind` | `string1.replace(2,4, "okay")` |
| `string substr(ind,n)` | Returns a string consisting of *n* characters extracted from the string, starting at index `ind`; if *n* is greater than the remaining number of characters, the rest of the string is used | `string2 = string1.substr(0,10)` |
| `void swap(str)` | Swaps characters in `str` with those in the first string | `string1.swap(string2)` |
| `string[ind]` (*Note*: This is standard array notation.) | Returns the character at index `ind`, without checking whether `ind` is a valid index | `string1[5]` |
| `=` | Assignment (also converts a C-string to a string) | `string1 = string` |
| `+` | Concatenates two strings | `string1 + string2` |
| `+=` | Concatenation and assignment | `string2 += string1` |
| `==` `!=` `<` `<=` `>` `>=` | Relational operators Return `true` if the relation is satisfied; otherwise, return `false` | `string1 == string2` `string1 <= string2` `string1 > string2` |

7) From all the methods above, one of the most commonly used is `length()`. It returns the number of characters in the string, which is referred to as the string's length.
8) For example, if the string referenced by `str` contains the value `"Have a good day."`, the value returned by the call `str.length()` is 16 (the quotation marks aren't considered part of the string).
9) Two string expressions can be compared for equality
    - each character in a string is stored in binary with the ASCII or Unicode code,
    - a blank is less than all letters and numbers,
    - letters of the alphabet are stored in order from A to Z,
    - digits are stored in order from 0 to 9,
    - digits are less than uppercase characters, which are followed by lowercase characters. Therefore, uppercase characters are mathematically less than lowercase characters.
    - when two strings are compared, their characters are compared a pair at a time (both first characters, then both second characters, and so on). If no differences are found, the strings

are equal; if a difference is found, the string with the first lower character is considered the smaller string.

10) Here are some examples:

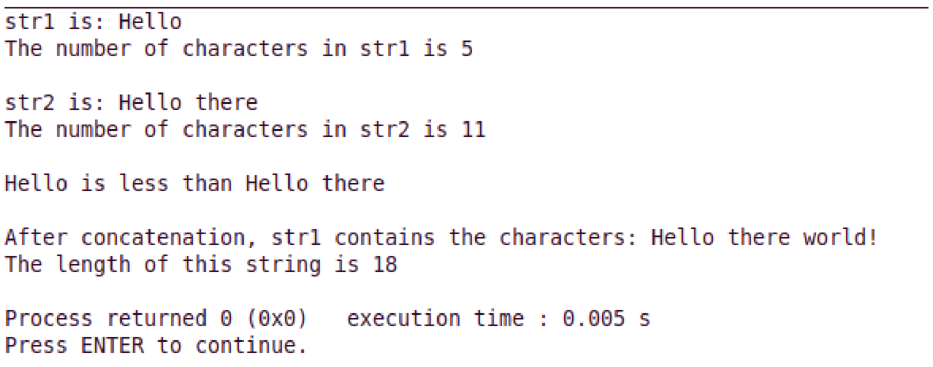


"Hello" is greater than "Good Bye" because the H in Hello is greater than the G in Good Bye.
"Hello" is less than "hello" because the H in Hello is less than the h in hello.
"SMITH" is greater than "JONES" because the S in SMITH is greater than the J in JONES.
"123" is greater than "1227" because the third character in 123, the 3, is greater than the third character in 1227, the 2.
"Behop" is greater than "Beehive" because the third character in Behop, the h, is greater than the third character in Beehive, the e.

11) Code 69: Use of `length()` and several relational expressions:

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
    string str1 = "Hello";
    string str2 = "Hello there";

    cout << "str1 is: " << str1 << endl;
    cout << "The number of characters in str1 is "
         << str1.length() << endl << endl;

    cout << "str2 is: " << str2 << endl;
    cout << "The number of characters in str2 is "
         << str2.length() << endl << endl;

    if (str1 < str2)
        cout << str1 << " is less than " << str2 << endl << endl;
    else if (str1 == str2)
        cout << str1 << " is equal to " << str2 << endl << endl;
    else
        cout << str1 << " is greater than " << str2 << endl << endl;

    str1 = str1 + " there world!";
    cout << "After concatenation, str1 contains the characters: "
         << str1 << endl;
    cout << "The length of this string is "
         << str1.length() << endl;
    return 0;
}
```

with output

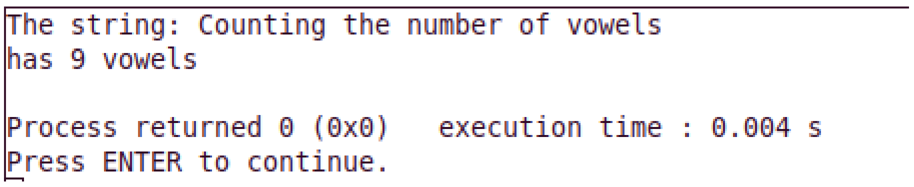


```
str1 is: Hello
The number of characters in str1 is 5

str2 is: Hello there
The number of characters in str2 is 11

Hello is less than Hello there

After concatenation, str1 contains the characters: Hello there world!
The length of this string is 18

Process returned 0 (0x0)   execution time : 0.005 s
Press ENTER to continue.
```

12) Code 70: Use of `at()` to retrieve separate characters in a string for counting vowels:

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
      string str = "Counting the number of vowels";
      int n = str.length();
      int vowelCount = 0;

      for (int i=0; i<n; i++)
      {
            switch(str.at(i))
            {
                  case 'a':
                  case 'e':
                  case 'i':
                  case 'o':
                  case 'u':
                  vowelCount++;
            }
      }

      cout << "The string: " << str << endl
           << "has " << vowelCount << " vowels." << endl;

      return 0;
}
```

which gives output



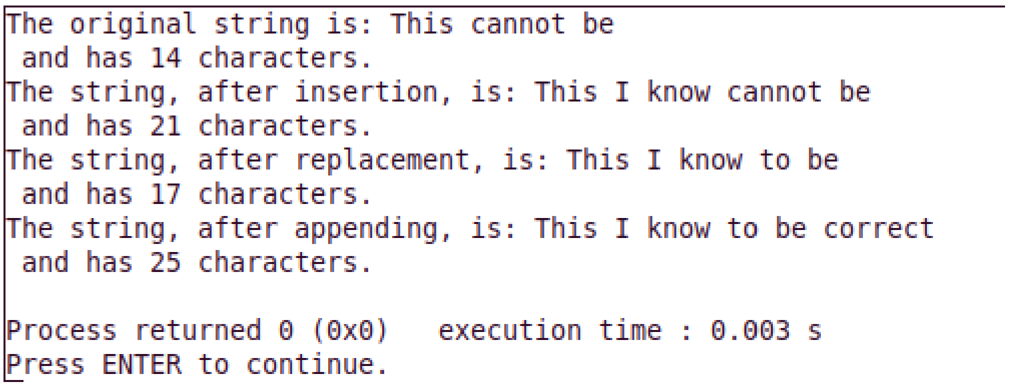
```
The string: Counting the number of vowels
has 9 vowels

Process returned 0 (0x0)   execution time : 0.004 s
Press ENTER to continue.
```

13) Here, the method `at()` in the `switch` statement retrieves the character at position `i` in the string. This character is then compared with five different character values.
14) The `switch` statement uses the fact that selected cases “drop through” in the absence of `break` statement. Therefore, all selected cases result in an increment to `vowelCount`.
15) Note also that we use the method `length()` to find the number of characters in the string.

16) Code 71: Inserting and replacing characters in a string

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
    string str = "This cannot be";
    cout << "The original string is: " << str << endl
         << " and has " << str.length() << " characters." << endl;

    // insert characters
    str.insert(5,"I know ");
    cout << "The string, after insertion, is: " << str << endl
         << " and has " << str.length() << " characters." << endl;

    // replace characters
    str.replace(12, 6, "to");
    cout << "The string, after replacement, is: " << str << endl
         << " and has " << str.length() << " characters." << endl;

    // append characters
    str = str + " correct";
    cout << "The string, after appending, is: " << str << endl
         << " and has " << str.length() << " characters." << endl;

    return 0;
}
```

which gives output

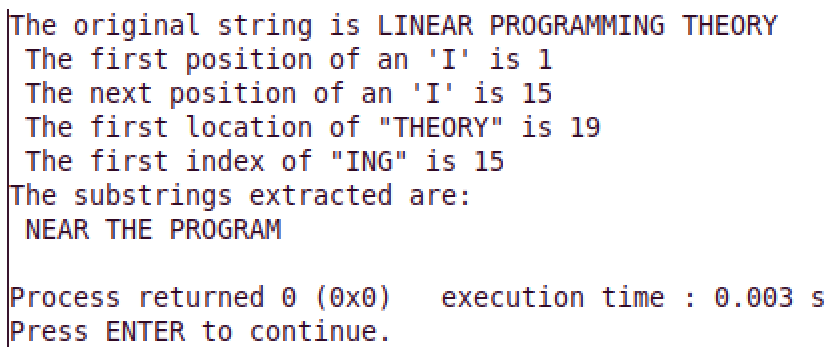


```
The original string is: This cannot be
 and has 14 characters.
The string, after insertion, is: This I know cannot be
 and has 21 characters.
The string, after replacement, is: This I know to be
 and has 17 characters.
The string, after appending, is: This I know to be correct
 and has 25 characters.

Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
```

17) Code 72: Locate specific characters in a string and create substrings (a substring is any sequence of characters contained in the original string)

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
    string str = "LINEAR PROGRAMMING THEORY";
    string s1, s2, s3;
    int j, k;

    cout << "The original string is " << str << endl;

    j = str.find('I');
    cout << " The first position of an 'i' is " << j << endl;

    k = str.find('I',(j+1));
    cout << " The next position of an 'i' is " << k << endl;

    j = str.find("THEORY");
    cout << " The first location of \"THEORY\" is " << j << endl;

    k = str.find("ING");
    cout << " The first index of \"ING\" is " << k << endl;

    // now extract three substrings
    s1 = str.substr(2,5);
    s2 = str.substr(19,3);
    s3 = str.substr(6,8);

    cout << "The substrings extracted are: " << endl
         << " " << s1 + s2+ s3 << endl;

    return 0;
}
```

which produces output:



```
The original string is LINEAR PROGRAMMING THEORY
 The first position of an 'I' is 1
 The next position of an 'I' is 15
 The first location of "THEORY" is 19
 The first index of "ING" is 15
The substrings extracted are:
 NEAR THE PROGRAM

Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
```

18) From the code above, it is clear that characters and sequences of characters can be located and extracted from a string with string class methods, which are `find()` and `substr()`, respectively.
19) The `find()` method is character-sensitive: replacing `'I'` with `'i'` will give different output.

**Character Manipulation Methods**

1) The header files `string` and `cctype` must be included in any program using these methods.
2) Examples of character manipulation methods:

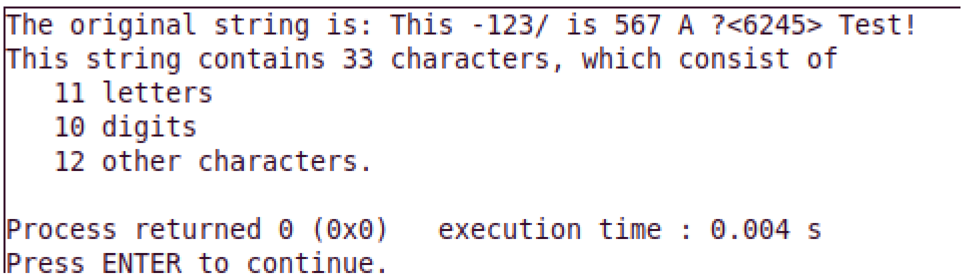
| Method Prototype | Description | Example |
|---|---|---|
| `int isalpha (charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a letter; otherwise, it returns a `false` (zero integer) | `isalpha('a')` |
| `int isalnum (charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a letter or a digit; otherwise, it returns a `false` (zero integer) | `isalnum(key)` |
| `int isupper (charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to an uppercase letter; otherwise, it returns a `false` (zero integer) | `isupper('a')` |
| `int islower (charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a lowercase letter; otherwise, it returns a `false` (zero integer) | `islower('a')` |
| `int isdigit (charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a digit (0 through 9); otherwise, it returns a `false` (zero integer) | `isdigit('a')` |
| `int isascii (charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to an ASCII character; otherwise, returns a `false` (zero integer) | `isascii('a')` |
| `int isspace (charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a space; otherwise, returns a `false` (zero integer) | `isspace(' ')` |
| `int isprint (charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a printable character; other-wise, returns a `false` (zero integer) | `isprint('a')` |
| `int isctrl (charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a control character; otherwise, it returns a `false` (zero integer) | `isctrl('a')` |
| `int ispunct (charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a punctuation character; other-wise, returns a `false` (zero integer) | `ispunct('!')` |
| `int isgraph (charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a printable character other than white space; otherwise, returns a `false` (zero integer) | `isgraph(' ')` |
| `int toupper (charExp)` | Returns the uppercase equivalent if `charExp` evaluates to a lowercase character; other-wise, returns the character code without modification | `toupper('a')` |
| `int tolower (charExp)` | Returns the lowercase equivalent if `charExp` evaluates to an uppercase character; otherwise, returns the character code without modification | `tolower('A')` |

3) Because all the methods above return a non-zero integer (interprated as a Boolean `true` value) when the character meets the condition and a zero integer (interprated as a Boolean `false` value) when the condition isn't met, these methods are typically used in an `if` statement.
4) The methods can also be used to process elements of C-strings (next topic).

5) Code 73: counting the number of letters, digits, and other characters in a string:

```
#include<iostream>
#include<string>
#include<cctype>
using namespace std;
int main()
{
      string str = "This -123/ is 567 A ?<6245> Test!";
      char nextChar;
      int nLetters = 0, nDigits = 0, nOthers = 0;

      // check each character in the string
      for (int i=0; i<str.length(); i++)
      {
            nextChar = str.at(i);
            if(isalpha(nextChar))
                  nLetters++;
            else if (isdigit(nextChar))
                  nDigits++;
            else
                  nOthers++;
      }

      cout << "The original string is: " << str
           << "\nThis string contains " << str.length()
           << " characters," << " which consist of" << endl
           << "    " << nLetters << " letters" << endl
           << "    " << nDigits << " digits" << endl
           << "    " << nOthers << " other characters." << endl;

      return 0;
}
```

which gives output

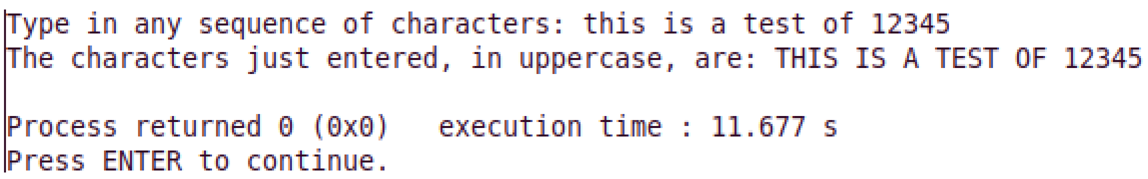


```
The original string is: This -123/ is 567 A ?<6245> Test!
This string contains 33 characters, which consist of
   11 letters
   10 digits
   12 other characters.

Process returned 0 (0x0)   execution time : 0.004 s
Press ENTER to continue.
```

6) Code 74:

```
#include<iostream>
#include<string>
#include<cctype>
using namespace std;
int main()
{
      string str;

      cout << "Type in any sequence of characters: ";
      getline(cin, str);

      // cycle through all elements of the string
      for (int i=0; i<str.length(); i++)
      str[i] = toupper(str[i]);

      cout << "The characters just entered, in uppercase, are: "
           << str << endl;

      return 0;
}
```

which gives output



```
Type in any sequence of characters: this is a test of 12345
The characters just entered, in uppercase, are: THIS IS A TEST OF 12345

Process returned 0 (0x0)   execution time : 11.677 s
Press ENTER to continue.
```

when given input "`this is a test of 12345`".

7) In the program above:

- the method `length()` is used to determine the end of the string.
- each element is accessed by using the array subscript notation `str[i]` rather than the `at()` method (they are interchangeable).
- the number `12345` is not changed since the method `toupper()` only alter lowercase characters and ignore other characters.

8) Other topics (but not covered in this course):

- exception handling
- exceptions and file checking
- input data validation

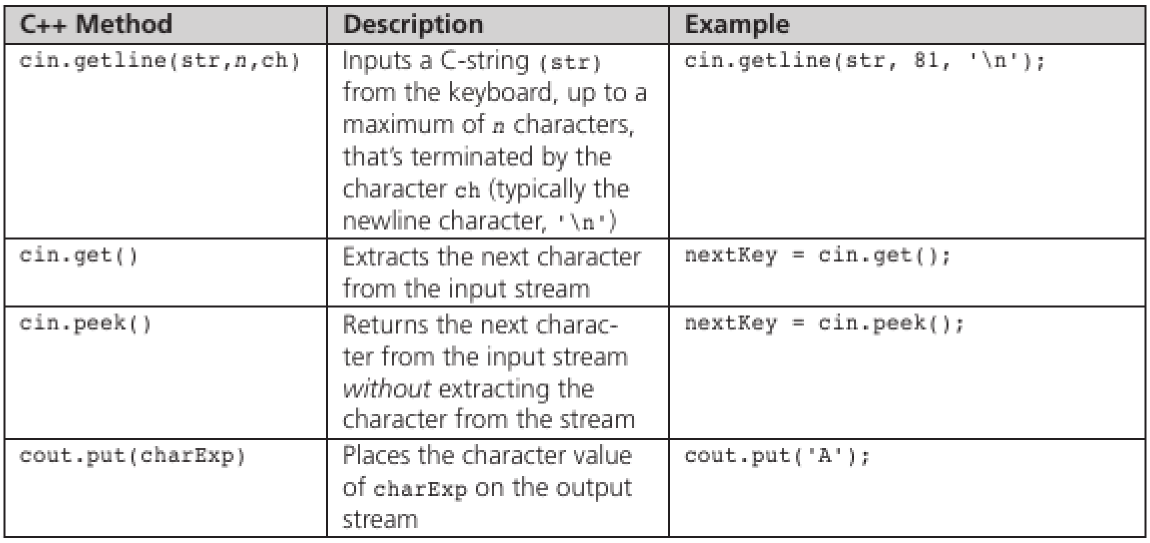
## II) C-STRING (string as character arrays)

**Background**

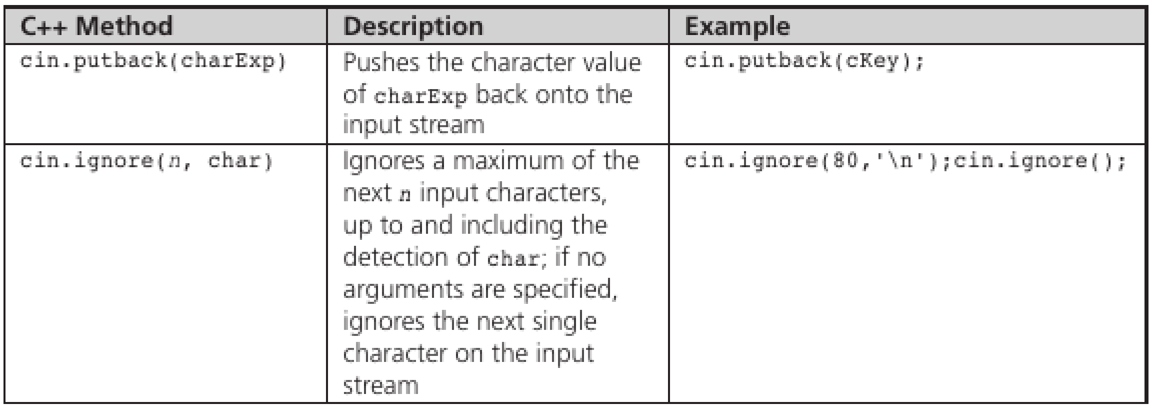
1. The `string` class (from previous topic) is a newer way to creating/using strings.
2. Using an array of characters to create a string (called as C-string/character string) is actually the original approach in C++.
3. In this approach, the array of character is terminated by a special end-of-string marker, called the `NULL` character, represented by the escape sequence '`\0`'.
4. Creating a C-string:
   - `char test[5] = “abcd”;`
   - `char test[]  = “abcd”;`
   - `char test[5] = {‘a’, ‘b’, ‘c’, ‘d’, ‘\0’};`
   - `char test[]  = {‘a’, ‘b’, ‘c’, ‘d’, ‘\0’};`
5. When a string (or string literal) is used for initialisation as in the first and second examples, the compiler automatically supplies the `NULL` character to the array.
6. Without the `‘\0’` in the third and fourth examples, `test` will just be a normal array of characters, not a string.
7. Because it is an array, a C-string can be input, manipulated, or output by using standard array-handling techniques, including subscript and pointer notation.

**C-String Input and Output**

1. These processes require standard input and output streams: `cin` and `cout,` and other class method defined by the `iostream` library.
2. For both
   - character-by-character
   - complete C-string
3. Amongst the string and character I/O methods are:



| C++ Method | Description | Example |
|---|---|---|
| `cin.getline(str,n,ch)` | Inputs a C-string (`str`) from the keyboard, up to a maximum of *n* characters, that's terminated by the character `ch` (typically the newline character, `'\n'`) | `cin.getline(str, 81, '\n');` |
| `cin.get()` | Extracts the next character from the input stream | `nextKey = cin.get();` |
| `cin.peek()` | Returns the next character from the input stream *without* extracting the character from the stream | `nextKey = cin.peek();` |
| `cout.put(charExp)` | Places the character value of `charExp` on the output stream | `cout.put('A');` |

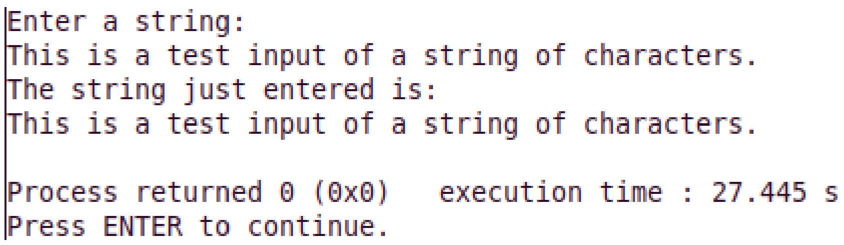


| C++ Method | Description | Example |
|---|---|---|
| `cin.putback(charExp)` | Pushes the character value of `charExp` back onto the input stream | `cin.putback(cKey);` |
| `cin.ignore(n, char)` | Ignores a maximum of the next *n* input characters, up to and including the detection of `char`; if no arguments are specified, ignores the next single character on the input stream | `cin.ignore(80,'\n');cin.ignore();` |

4. The methods `cin.getline()`, `cin.get()`, and `cin.peek()` are provided for input. They are not the same as the methods with the same names defined for the `string` class.
5. The character output functions `put()` and `putback()`, however are the same as those for the `string` class.
6. Code 75: Using `cin.getline` and `cout` to input and output a string:

```
#include <iostream>
using namespace std;
int main()
{
      const int MAXCHARS = 81;
      char message[MAXCHARS]; // an array of chars with enough storage
                              // for a complete string (line)
      cout << "Enter a string:\n";
      cin.getline(message,MAXCHARS,'\n');

      cout << "The string just entered is:\n"
           << message << endl;

      return 0;
}
```

which gives output



```
Enter a string:
This is a test input of a string of characters.
The string just entered is:
This is a test input of a string of characters.

Process returned 0 (0x0)   execution time : 27.445 s
Press ENTER to continue.
```

when supplied with input "`This is a test input of a string of characters`"

7. In the code above:
   - `cin.getline()` method continuously accepts and stores characters typed at the keyboard into the character array named `message` until 80 characters are entered (the 81st character is then used to store the end-of string `NULL` character, `'\0'`) or the Enter key is detected.
   - Pressing the Enter key generates a newline character, `'\n'` which `cin.getline()` interprets as the end-of-line entry.

- All the characters `cin.getline()` encounters, except the newline character, are stored in the `message` array. Before returning, `cin.getline()` appends a `NULL` character, `'\0'`, to the stored set of characters, as shown below.

$$\begin{tikzcd}\textit{characters}\ \texttt{\textbackslash n} \arrow[r] & \boxed{\texttt{cin.getline()}} \arrow[r] & \textit{characters}\ \texttt{\textbackslash 0}\end{tikzcd}$$

- The `cin` object can't be used in place of `cin.getline()` for C-string input because it stops reading characters when it encounters a blank space or a newline character.
- The `cin.getline()` method has this syntax:

```
cin.getline(str, terminatingLength, terminatingChar)
```

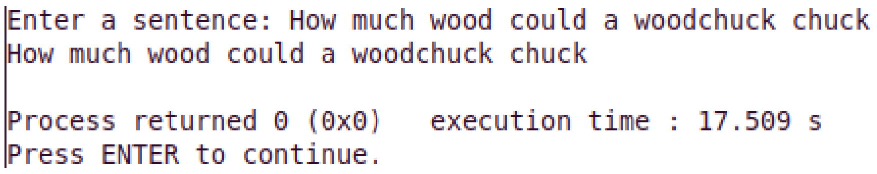
- The third input, `terminatingChar`, is optional. If it is omitted, the default terminating character is the newline (`'\n'`) character.
- Interestingly, we can display the whole array of characters by using `cout` without a `for` loop.

**<u>C-String Processing</u>**

1. C-string can be manipulated with standard library functions, or as subscripted array variables (pointers can also be used).
2. For now, concentrate on processing a C-string in a character-by-character fashion with subscripts. The library functions are available for use, but will be discussed soon.
3. Code 76: copying the contents of a C-string to another C-string

```
#include <iostream>
using namespace std;
void strcopy(char [], char []);   // function prototype
int main()
{
      const int MAXCHARS = 81;
      char str1[MAXCHARS], str2[MAXCHARS];

      cout << "Enter a sentence: " ;
      cin.getline(str1,MAXCHARS);
      strcopy(str1,str2);              // pass two array addresses
      cout << str2 << endl;

      return 0;
}

void strcopy(char s1[], char s2[])
{
      int i = 0;
      while (s1[i] != '\0')
      {
            s2[i] = s1[i];
            i++;
      }
      s2[i] = '\0';
}
```

which gives output

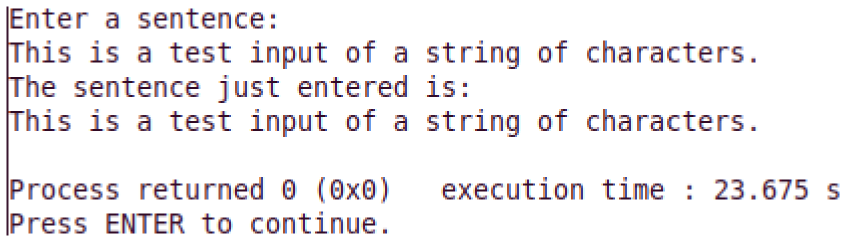


```
Enter a sentence: How much wood could a woodchuck chuck
How much wood could a woodchuck chuck

Process returned 0 (0x0)   execution time : 17.509 s
Press ENTER to continue.
```

4. The code above can actually be shortened considerably and written more compactly.
5. However, it does illustrates the main feature of C-string manipulation:
   - accesing array elements by using subscripts (can also use pointers)
   - using the end-of-string `NULL` character to determine when to stop processing.
6. Character-by-character input
   - the purpose is to use `cin.get()` to accept a string one character at a time.
   - Code 77:

```
#include <iostream>
using namespace std;
int main()
{
      const int MAXCHARS = 81;
      char str[MAXCHARS], c;

      cout << "Enter a sentence: \n";

      int i=0;
      while(i<MAXCHARS && (c=cin.get()) !='\n')
      {
            str[i]=c;
            i++;
      }
      str[i] = '\0';

      cout << "The sentence just entered is:\n";
      cout << str << endl;

      return 0;
}
```

which gives output



```
Enter a sentence:
This is a test input of a string of characters.
The sentence just entered is:
This is a test input of a string of characters.

Process returned 0 (0x0)   execution time : 23.675 s
Press ENTER to continue.
```

- In this code above:
  - ➢ each entered character is stored correctly in the array, provided the number of characters entered is less than 81 and the character returned by `cin.get()` isn't the newline character.
  - ➢ The parentheses surrounding the expression `c = cin.get()` are necessary to assign the character returned by `cin.get()` to the variable `c` before comparing it with the newline escape sequence.

**<u>C-string manipulation functions</u>**

1) The declarations for these functions are in the standard header file cstring and must be included in a C++ program before the functions are called.
2) Below is the list of the more common C-string library functions, which are called in the same manner as all C++ functions.

| Name | Description | Example |
|---|---|---|
| `strcpy(stringVar, stringExp)` | Copies `stringExp` to `stringVar`, including the `'\0'`. | `strcpy(test, "efgh")` |
| `strcat(stringVar, stringExp)` | Appends `stringExp` to the end of the string value contained in `stringVar`. | `strcat(test, "there")` |
| `strlen(stringExp)` | Returns the length of the string. Does not include the `'\0'` in the length count. | `strlen("Hello World!")` |

| Name | Description | Example |
|---|---|---|
| `strcmp(stringExp1,stringExp2)` | Compares `stringExp1` with `stringExp2`. Returns a negative integer if `stringExp1 < stringExp2`, 0 if `stringExp1 == stringExp2`, and a positive integer if `stringExp1 > stringExp2`. | `strcmp("Bebop","Beehive")` |
| `strncpy(stringVar,stringExp,n)` | Copies at most *n* characters of `stringExp` to `stringVar`. If `stringExp` has fewer than *n* characters, it pads `stringVar` with '\0's. | `strncpy(str1, str2, 5)` |
| `strncmp(stringExp1,stringExp2, n)` | Compares at most *n* characters of `stringExp1` with `stringExp2`. Returns the same values as `strcmp()` based on the number of characters compared. | `strncmp("Bebop",Beehive",2)` |
| `strchr(stringExp, character)` | Locates the first occurrence of the character in the string. Returns the address of the character. | `strchr("Hello", 'l')` |
| `strtok(string1, character)` | Parses `string1` into tokens. Returns the next sequence of characters contained in `string1`, up to but not including the delimiter character `character`. | `strtok("Hello there World!",' ')` |

3) The first four functions listed above are used most often.
4) For example, in the function call `strcpy(str1 "Hello World!")`, the source string literal `"Hello World!"` is copied into the destination C-string variable `str1`. However it is the programmer's responsibility to ensure that `str1` is large enough to contain the source C-string.

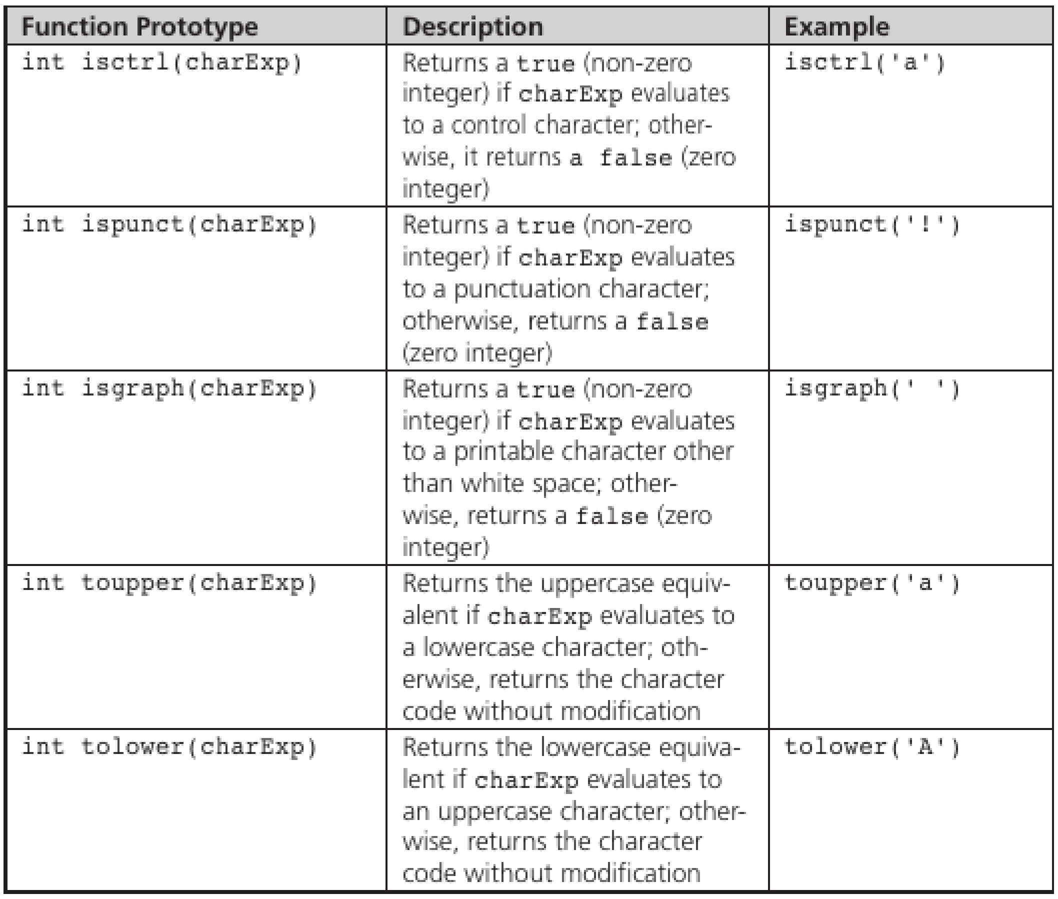
5) The function `strlen()` returns the number of characters in its C-string parameter but doesn't include the terminating `NULL` character in the count. For example, the value returned by the function call `strlen("Hello World!")` is 12.

## Character manipulation functions

1) In addition to C-string manipulation functions, all C++ compilers include the character-handling functions.
2) These functions are exactly the same as the ones used to manipulate characters from string objects in `string` class discussed in previous topic.
3) Note that we need the header files `string` and `cctype` to use this function.

| Function Prototype | Description | Example |
|---|---|---|
| `int isalpha(charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a letter; otherwise, it returns a `false` (zero integer) | `isalpha('a')` |

| Function Prototype | Description | Example |
|---|---|---|
| `int isalnum(charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a letter or a digit; otherwise, it returns a `false` (zero integer) | `isalnum(key);` |
| `int isupper(charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to an uppercase letter; otherwise, it returns a `false` (zero integer) | `isupper('a')` |
| `int islower(charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a lowercase letter; otherwise, it returns a `false` (zero integer) | `islower('a')` |
| `int isdigit(charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a digit (0 through 9); otherwise, it returns a `false` (zero integer) | `isdigit('5')` |
| `int isascii(charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to an ASCII character; otherwise, returns a `false` (zero integer) | `isascii('a')` |
| `int isspace(charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a space; otherwise, returns a `false` (zero integer) | `isspace(' ')` |
| `int isprint(charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a printable character; otherwise, returns a `false` (zero integer) | `isprint('a')` |



| Function Prototype | Description | Example |
|---|---|---|
| `int isctrl(charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a control character; otherwise, it returns a `false` (zero integer) | `isctrl('a')` |
| `int ispunct(charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a punctuation character; otherwise, returns a `false` (zero integer) | `ispunct('!')` |
| `int isgraph(charExp)` | Returns a `true` (non-zero integer) if `charExp` evaluates to a printable character other than white space; otherwise, returns a `false` (zero integer) | `isgraph(' ')` |
| `int toupper(charExp)` | Returns the uppercase equivalent if `charExp` evaluates to a lowercase character; otherwise, returns the character code without modification | `toupper('a')` |
| `int tolower(charExp)` | Returns the lowercase equivalent if `charExp` evaluates to an uppercase character; otherwise, returns the character code without modification | `tolower('A')` |

Other topics (but not covered in this course):

- Conversion Functions: used to convert C-strings to and from integer and double-precision data types.
- C-String Definitions and Pointer Arrays