Cstring

```
#include <cstring>
```

A C-style string is a character array that ends with the null character.

```
char greeting[20] = "Hello, World \0";
```

Getline Example:

```
 char greeting[15], name[10], other[20];
 cin.getline(greeting,15);
 cin.get(name,10,'.'); // reads up to'.'
 cin.getline(other,20);
```

Input:

```
 Hello, World
 Joe Smith.  He says hello.
```

Answers:

```
 greeting:  "Hello, World"
 name:      "Joe Smith"
 other:     ".  He says hello."
```

---

int strlen(const char str[]);

```
 char phrase[30] = "Hello, World";
 cout<<strlen(phrase);         // shows 12
```

---

char* strcpy(char str1[], const char str2[]);

copies the second string into the first string from str[0].

---

char* strcat(char str1[], const char str2[]);

concatenates the second one onto the first.

```
 char buffer[80] = "Dog";
 char word[] = "food";
 strcat(buffer, word); //buffer="Dogfood"
```

---

int strcmp(const char str1[], const char str2[]);

```
//get a negative number, if str1 < str2
//get a positive number, if str1 > str2
//get 0 if they are equal
//(uppercase before lowercase in ascii)

 char buffer[80];
 char word[11] = "applesauce";
 char bigword[] =
"antidisestablishmentarianism";

  strncpy(buffer, word, 5); //"apple"
  strncat(buffer, " piecemeal", 4);
// buffer now stores "apple pie"
  strncmp(buffer, "apple", 5); // get 0
  strncpy(word, bigword, 10);
// word is now "antidisest", word only
had 11 slots!
```

String

```
#include <string>
 if (s1 == s2)
 cout << "The strings are the same";

 if (s1 < s2)
 cout << "s1 comes first
lexicograpically";
```

The ordering on strings is a lexicographical ordering, which goes by ASCII values of the characters

```
"apple" < "apply"
"apple" > "Apply"
"apple" > "Zebra"
```

`size()` -- returns the length of the string

`capacity()` -- returns the current allocated size of the string object (allocation might be larger than current usage, which is the length)

`resize(X, CH)` -- changes the string's allocated size to X. If X is bigger than the currently stored string, the extra space at the end is filled in with the character CH

`clear()` -- delete the contents of the string. Reset it to an empty string

`empty()` -- return true if the string is currently empty, false otherwise

`at(X)` -- return the character at position X in the string. Similar to using the [] operator

`substr(X, Y)` -- returns a copy of the substring (i.e. portion of the original string) that starts at index X and is Y characters long

`substr(X)` -- returns a copy of the substring, starting at index X of the original string and going to the end

Examples:

```
   string s1 = "Greetings, earthling";
   string s2 = s1.substr(11,5);
      // s2 is now "earth"
   string s3 = s1.substr(4);        // s2
is now "tings, earthling"
```

`append(str2)` -- appends str2 (a string or a c-string)

`append(str2, Y)` -- appends the first Y characters from str2 (a string or char array)

`append(str2, X, Y)` -- appends Y characters from str2, starting at index X

`append(X, CH)` -- appends X copies of the character CH

Cctype

These return the ascii value

```
int tolower(int c)
int isdigit(int c)
int isalpha(int c)
int isalnum(int c) - digit or a letter?
int islower(int c)
int isupper(int c)
int isspace(int c) - white space
```

Automatic Type Conversions

Can go from "smaller" to "larger" types.

```
char -> short -> int -> long -> float ->
double -> long double
 int          i1, i2;
 double       d1, d2;
 char         c1;
 d1 = i1;       // legal.
 c1 = i1;       // illegal
 i1 = d1;       // illegal
 i1 = c1;       // legal
 d2 = d1 + i2;  // get double
 d2 = d1 / i2;  // with floating point
cstring
 c1 = (char)i2;
 i1 = (int)d2;
c++
 c1 = static_cast<char>(i2);
 i1 = static_cast<int>(d2);
```

Selection Statements

```
if (expression){
      statement;}
else
      statement;
----------
switch (expression)
{
    case const1/'a':
        statements
    case const2/'b':
        statements
        …
    default:            // optional label
        statements }
//scan from the case to the end, break to
stop
----------
test_expression ? true_expression :
false_expression
```

Example:

```
(x < 0 ? value = 10 : value = 20);
value = (x < 0 ? 10 : 20); //the same
```

Control Sructures – Repetition

```
// while loop format
 while (expression){
    statement;}

 // do-while loop format
 do
 {
    statement;
 } while (expression);
----------
 for (initialCondition; testExpression;
iterativeStatement) {
Statement;}

 if (statement) continue;
//jump to next iteration
 if (experession) break;
//break the loop
```

Functions

```
 #include <iostream>   // I/O routines
 #include <cmath>      // math functions
 #include <cstdlib>    // C functions

//Declaring a Function
 double Sum(double x, float y, int z);
 bool InOrder(int x, int y, int z);
 int DoTask(char letter, int num);

return-type function-name(parameter-list)
 {
    body(declarations and statements)
    return expression;}
```

Pass By Reference vs. Pass By Value

```
 int n = 5;  //a variable, n
 int & r = n; //r is a reference to n
```

Pass by value: a copy will be made.

```
 void Func1(int x, double y)
 {
   x = 12; //will not affect the caller
   y = 20.5; //change LOCAL x and y
 }
```

Pass by reference: sends back a reference to the original.

```
 void Func2(int& x, double& y)
 {
   x = 12; // WILL change the original x
   y = 20.5;
 }
int Task1(int x, double y); // by value
int& Task2(int x, double y); // by ref
```