04: FUNCTION

Programming Technique I
(SCSJ1013)

Modular Programming

- **Modular programming**: breaking a program up into smaller, manageable functions or modules
- **Function**: a collection of statements to perform a task
- Motivation for modular programming:
  - Improves maintainability of programs
  - Simplifies the process of writing programs

This program has one long, complex function containing all of the statements necessary to solve a problem.

```
int main()
{
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
}
```

In this program the problem has been divided into smaller problems, each of which is handled by a separate function.

```
int main()
{
   statement;
   statement;
   statement;
}
```

main function

```
void function2()
{
   statement;
   statement;
   statement;
}
```

function 2

```
void function3()
{
   statement;
   statement;
   statement;
}
```

function 3

```
void function4()
{
   statement;
   statement;
   statement;
}
```

function 4

Function

- A **collection of statements** that performs a specific task.
- Commonly used to **break a problem** down into small manageable pieces.
- In C++, there are 2 types of function:
  - Library functions
  - User-defined functions

Library Functions

- “Built-in" functions that **come with the compiler**.
- The source code (definition) for library functions does NOT appear in your program.
- To use a library function, you simply need to **include the proper header file** and know the name of the function that you wish to use.
  - **#include** *compiler directive*

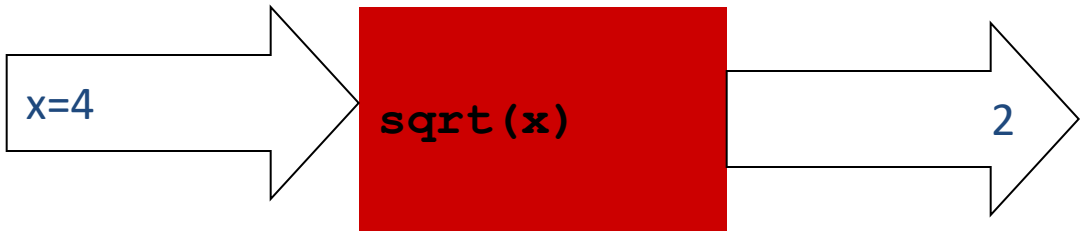
Library Functions (cont.)

- Libraries under discussion at this time:

Compiler directive	Purpose
<ctype>	Character classification and conversion
<cmath>	Math functions
<stdlib>	Data conversion
<time>	Time functions

Library Functions (cont.)

- A collection of specialized functions.
- C++ promotes code reuse with predefined classes and functions in the standard library
- The functions work as a black box:



Math Functions

- Required header: #include <cmath>
- Example functions

Function	Purpose
abs(x)	returns the absolute value of an integer.
pow(x,y)	calculates x to the power of y. If x is negative, y must be an integer. If x is zero, y must be a positive integer.
pow10(x)	calculates 10 to the power of x.
sqrt(x)	calculates the positive square root of x. (x is >=0)

Library Functions: Example 1

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    double area, radius;

    cout<< "This program calculates the area of a
    circle.\n";
    cout<<"What is the radius of the circle? ";
    cin>>radius;
    area=3.14159 * pow(radius,2.0);
    cout<<"The area is " << area <<endl;
    system ("pause");
    return 0;
}
```

Library Functions: Example 2

```
#include <iostream>
#include <cmath>
using namespace std;
main(){
    int nom1,nom2, result;
    cout<<"Enter two numbers";
    cin>>nom1>>nom2;
    if ((nom1<0)||(nom2<0))
    {   cout<<"negative number/s";
    }
    else
    {
       result= sqrt(nom1 + nom2);
       cout<<"The square root of "<< nom1+nom2 << "is"
    << result;}
}
```

More Mathematical Functions

Function	Standard Library	Purpose: Example	Argument(s)	Result
`abs(x)`	`<cstdlib>`	Returns the absolute value of its integer argument: if `x` is `-5`, `abs(x)` is `5`	`int`	`int`
`ceil(x)`	`<cmath>`	Returns the smallest integral value that is not less than `x`: if `x` is `45.23`, `ceil(x)` is `46.0`	`double`	`double`
`cos(x)`	`<cmath>`	Returns the cosine of angle `x`: if `x` is `0.0`, `cos(x)` is `1.0`	`double` (radians)	`double`
`exp(x)`	`<cmath>`	Returns e^x where e = `2.71828...`: if `x` is `1.0`, `exp(x)` is `2.71828`	`double`	`double`
`fabs(x)`	`<cmath>`	Returns the absolute value of its type `double` argument: if `x` is `-8.432`, `fabs(x)` is `8.432`	`double`	`double`
`floor(x)`	`<cmath>`	Returns the largest integral value that is not greater than `x`: if `x` is `45.23`, `floor(x)` is `45.0`	`double`	`double`

More Mathematical Functions

`log(x)`	`<cmath>`	Returns the natural logarithm of x for `x > 0.0`: if x is `2.71828`, `log(x)` is `1.0`	`double`	`double`
`log10(x)`	`<cmath>`	Returns the base-10 logarithm of x for `x > 0.0`: if x is `100.0`, `log10(x)` is `2.0`	`double`	`double`
`pow(x, y)`	`<cmath>`	Returns x^y. If x is negative, y must be integral: if x is `0.16` and y is `0.5`, `pow(x, y)` is `0.4`	`double, double`	`double`
`sin(x)`	`<cmath>`	Returns the sine of angle x: if x is `1.5708`, `sin(x)` is `1.0`	`double` (radians)	`double`
`sqrt(x)`	`<cmath>`	Returns the non-negative square root of x ($\sqrt{x}$) for `x ≥ 0.0`: if x is `2.25`, `sqrt(x)` is `1.5`	`double`	`double`
`tan(x)`	`<cmath>`	Returns the tangent of angle x: if x is `0.0`, `tan(x)` is `0.0`	`double` (radians)	`double`

More Mathematical Functions

- Require `cmath` header file
- Take `double` as input, return a `double`
- Commonly used functions:

`sin`	Sine
`cos`	Cosine
`tan`	Tangent
`sqrt`	Square root
`log`	Natural (e) log
`abs`	Absolute value (takes and returns an int)

More Mathematical Functions

- These require `cstdlib` header file
- `rand`(): returns a random number (`int`) between 0 and the largest int the compute holds. Yields same sequence of numbers each time program is run.
- `srand`(`x`): initializes random number generator with `unsigned int x`

General Purpose Library

- These require `cstdlib` header file
- `rand()`: returns a random number (`int`) between `0` and the largest int the compute holds. Yields same sequence of numbers each time program is run.
- `srand(x)`: initializes random number generator with `unsigned int x`

Character Manipulation

- The C++ library provides several functions for testing characters.
- To use these functions, you must include the `cctype` header file.

FUNCTION	MEANING
isalpha	true if arg. is a letter, false otherwise
isalnum	true if arg. is a letter or digit, false otherwise
isdigit	true if arg. is a digit 0-9, false otherwise
islower	true if arg. is lowercase letter, false otherwise
isprint	true if arg. is a printable character, false otherwise
ispunct	true if arg. is a punctuation character, false otherwise
isupper	true if arg. is an uppercase letter, false otherwise
isspace	true if arg. is a whitespace character, false otherwise

Example – Character Testing

```
#include <iostream>
#include <cctype>
using namespace std;
int main()
{
    char input;
    cout<<"Enter any character: ";
    cin.get(input);
    if (isalpha(input)){
        cout.put(input);
        cout<<"It is an alphabet";}
    if (isdigit(input))
        cout<<"It is a digit";
    if (islower(input))
        cout<<"The letter entered is lowercase";
    if (isupper(input))
        cout<<"The letter entered is uppercase";
    return 0;
}
```

Exercise 01

- Write a program with if statements that will display the word "digit" **`if`** the variable **`ch`** contains numeric data, or display the word "letter" if the variable **`ch`** contains alphabet data. Otherwise, it should display "special character"

Character Case Conversion

- Require `cctype` header file
- Functions:

  `toupper`: if `char` argument is lowercase letter, return uppercase equivalent; otherwise, return input unchanged

```
char ch1 = 'H';
char ch2 = 'e';
char ch3 = '!';
cout << toupper(ch1);  // displays 'H'
cout << toupper(ch2);  // displays 'E'
cout << toupper(ch3);  // displays '!'
```

Character Case Conversion

- Functions:

  `tolower`: if `char` argument is uppercase letter, return lowercase equivalent; otherwise, return input unchanged

```
char ch1 = 'H';
char ch2 = 'e';
char ch3 = '!';
cout << tolower(ch1);  // displays 'h'
cout << tolower(ch2);  // displays 'e'
cout << tolower(ch3);  // displays '!'
```

Example – Character Case Conversion

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

int main()
{
    char input[15];
    cout<<"Enter a name ";
    cin>>input;
    for(int i=0;input[i] != '\0';i++)
    input[i]=toupper(input[i]);
    cout<<"The name in upper case is:" << input;
    return 0;
}
```

Character Manipulating Functions

Display 9.3 **Some Functions in <cctype> *(part 1 of 2)***

FUNCTION	DESCRIPTION	EXAMPLE

`toupper(Char_Exp)`	Returns the uppercase version of *Char_Exp* (as a value of type `int`).	`char c = toupper('a');`
		`cout << c;`
		Outputs: A

`tolower(Char_Exp)`	Returns the lowercase version of *Char_Exp* (as a value of type `int`).	`char c = tolower('A');`
		`cout << c;`
		Outputs: a

`isupper(Char_Exp)`	Returns `true` provided *Char_Exp* is an uppercase letter; otherwise, returns `false`.	`if (isupper(c))`
		`  cout << "Is uppercase.";`
		`else`
		`  cout << "Is not uppercase.";`

`islower(Char_Exp)`	Returns `true` provided *Char_Exp* is a lowercase letter; otherwise, returns `false`.	`char c = 'a';`
		`if (islower(c))`
		`  cout << c << " is lowercase.";`
		Outputs: a is lowercase.

`isalpha(Char_Exp)`	Returns `true` provided *Char_Exp* is a letter of the alphabet; otherwise, returns `false`.	`char c = '$';`
		`if (isalpha(c))`
		`  cout << "Is a letter.";`
		`else`
		`  cout << "Is not a letter.";`
		Outputs: Is not a letter.

Display 9.3 **Some Functions in <cctype>** ***(part 2 of 2)***

FUNCTION	DESCRIPTION	EXAMPLE

`isdigit(Char_Exp)`	Returns `true` provided *Char_Exp* is one of the digits '0' through '9'; otherwise, returns `false`.	`if (isdigit('3'))`
		`  cout << "It's a digit.";`
		`else`
		`  cout << "It's not a digit.";`
		Outputs: It's a digit.

`isalnum(Char_Exp)`	Returns `true` provided *Char_Exp* is either a letter or a digit; otherwise, returns `false`.	`if (isalnum('3') && isalnum('a'))`
		`  cout << "Both alphanumeric.";`
		`else`
		`  cout << "One or more are not.";`
		Outputs: Both alphanumeric.

`isspace(Char_Exp)`	Returns `true` provided *Char_Exp* is a whitespace character, such as the blank or newline character; otherwise, returns `false`.	`//Skips over one "word" and sets c`
		`//equal to the first whitespace`
		`//character after the "word":`
		`do`
		`{`
		`  cin.get(c);`
		`} while (! isspace(c));`

`ispunct(Char_Exp)`	Returns `true` provided *Char_Exp* is a printing character other than whitespace, a digit, or a letter; otherwise, returns `false`.	`if (ispunct('?'))`
		`  cout << "Is punctuation.";`
		`else`
		`  cout << "Not punctuation.";`

`isprint(Char_Exp)`	Returns `true` provided *Char_Exp* is a printing character; otherwise, returns `false`.	
`isgraph(Char_Exp)`	Returns `true` provided *Char_Exp* is a printing character other than whitespace; otherwise, returns `false`.	
`isctrl(Char_Exp)`	Returns `true` provided *Char_Exp* is a control character; otherwise, returns `false`.	

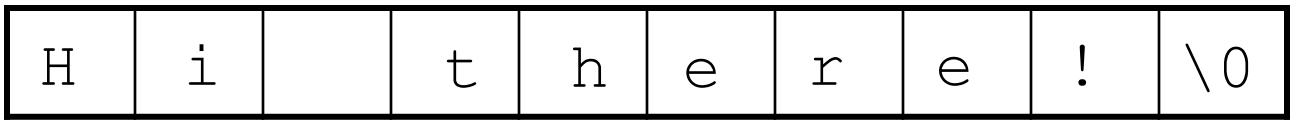
Exercise 02

- Write a program to toggle the contents of a string character from lower to upper case or vice verse.

Review of the Internal Storage of C-Strings

- **C-string**: sequence of characters stored in adjacent memory locations and terminated by `NULL` character
- **String literal** (**string constant**): sequence of characters enclosed in double quotes " " :

```
"Hi there!"
```



Review of the Internal Storage of C-Strings

- Array of `chars` can be used to define storage for string:

```
const int SIZE = 20;
char city[SIZE];
```

- Leave room for `NULL` at end
- Can enter a value using `cin` or `>>`
  - Input is whitespace-terminated
  - No check to see if enough space
- For input containing whitespace, and to control amount of input, use `cin.getline()`

C-Strings Manipulation Functions

- The C++ library has numerous functions for handling C-strings.
- Requires `cstring` header file be included.
- Functions take one or more C-strings as arguments. Can use:
  - C-string name
  - pointer to C-string
  - literal string

Functions for C-Strings

- Functions:
  - `strlen(str)`: returns length of C-string `str`

```
char city[SIZE] = "Missoula";
cout << strlen(city); // prints 8
```

  - `strcat(str1, str2)`: appends str2 to the end of str1

```
char location[SIZE] = "Missoula, ";
char state[3] = "MT";
strcat(location, state);
// location now has "Missoula, MT"
```

Function for C-Strings

- Functions:
  - `strcpy(str1, str2)`: copies `str2` to `str1`

```
const int SIZE = 20;
char fname[SIZE] = "Maureen",
name[SIZE];
strcpy(name, fname);
```

Note: `strcat` and `strcpy` perform no bounds checking to determine if there is enough space in receiving character array to hold the string it is being assigned.

Function for C-Strings

- Functions:
  - `strstr(str1, str2)`: finds the first occurrence of `str2` in `str1`. Returns a pointer to match, or `NULL` if no match.

```
char river[] = "Wabash";
char word[] = "aba";
cout << strstr(state, word);
// displays "abash"
```

Comparing C-Strings

- Also cannot use operator ==

```
char aString[10] = "Hello";
char anotherString[10] = "Goodbye";
aString == anotherString; // NOT allowed!
```

- Must use library function again:

```
if (strcmp(aString, anotherString))
    cout << "Strings NOT same.";
else
    cout << "Strings are same.";
```

Working with C-Strings - Example

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{  char reply;
    char garment[]="overcoat";
    cout<<"Is it raining outside? Answer y/n\n";
    cin>>reply;
    if(reply=='y')
        strcpy(garment,"raincoat");
  cout<<"before you go out today take your "<<garment;
    return 0;
}
```

Predefined C-String Functions

Display 9.1 **Some Predefined C-String Functions in <cstring> *(part 1 of 2)***

FUNCTION	DESCRIPTION	CAUTIONS
`strcpy(`*Target_String_Var*`,` *Src_String*`)`	Copies the C-string value *Src_String* into the C-string variable *Target_String_Var*.	Does not check to make sure *Target_String_Var* is large enough to hold the value *Src_String*.
`strcpy(`*Target_String_Var*`,` *Src_String*`,` *Limit*`)`	The same as the two-argument `strcpy` except that at most *Limit* characters are copied.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcpy`. Not implemented in all versions of C++.
`strcat(`*Target_String_Var*`,` *Src_String*`)`	Concatenates the C-string value *Src_String* onto the end of the C-string in the C-string variable *Target_String_Var*.	Does not check to see that *Target_String_Var* is large enough to hold the result of the concatenation.
`strcat(`*Target_String_Var*`,` *Src_String*`,` *Limit*`)`	The same as the two argument `strcat` except that at most *Limit* characters are appended.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcat`. Not implemented in all versions of C++.

Predefined C-String Functions

Display 9.1 **Some Predefined C-String Functions in <cstring> *(part 2 of 2)***

FUNCTION	DESCRIPTION	CAUTIONS
`strlen(`*Src_String*`)`	Returns an integer equal to the length of *Src_String*. (The null character, `'\0'`, is not counted in the length.)	
`strcmp(`*String_1*`,` *String_2*`)`	Returns `0` if *String_1* and *String_2* are the same. Returns a value < 0 if *String_1* is less than *String_2*. Returns a value > 0 if *String_1* is greater than *String_2* (that is, returns a nonzero value if *String_1* and *String_2* are different). The order is lexicographic.	If *String_1* equals *String_2*, this function returns `0`, which converts to `false`. Note that this is the reverse of what you might expect it to return when the strings are equal.
`strcmp(`*String_1*`,` *String_2*`,` *Limit*`)`	The same as the two-argument `strcat` except that at most *Limit* characters are compared.	If *Limit* is chosen carefully, this is safer than the two-argument version of `strcmp`. Not implemented in all versions of C++.

String/Numeric Conversion Functions

- These functions convert between string and numeric forms of numbers
- Need to include `cstdlib` header file

FUNCTION	PARAMETER	ACTION
atoi	C-string	converts C-string to an int value, returns the value
atol	C-string	converts C-string to a long value, returns the value
atof	C-string	converts C-string to a double value, returns the value
itoa	int,C-string, int	converts 1st int parameter to a C-string, stores it in 2nd parameter. 3rd parameter is base of converted value

String/Numeric Conversion Functions

- `atoi` converts alphanumeric to int
- `atol` converts alphanumeric to long
- `atof` converts a numeric string to a double

- if C-string being converted contains non-digits, results are undefined
  - function may return result of conversion up to first non-digit
  - function may return 0

String/Numeric Conversion Functions

- Examples:
  - `int number;`
  - `long lnumber;`
  - `double dnumber;`
  - `number = atoi("57");`
  - `lnumber = atol("50000");`
  - `dnumber = atof(“590.55”);`

String/Numeric Conversion Functions

- **itoa** converts an **i**nt **to** an **a**lphanumeric string
- Allows user to specify the base of conversion

```
itoa(int num, char numStr, int base)
```

  - **num** : number to convert
  - **numStr**: array to hold resulting string
  - **base**: base of conversion
- Example: To convert the number 1200 to a hexadecimal string
  - `char numStr[10];`
  - `itoa(1200, numStr, 16);`