International University of Technology Twintech
جامعة تونتك الدولية للتكنولوجيا

Faculty of Computer Science and Information Technology
Department of Business Information Technology
Sana'a - Yemen

Lab Manual # 9&10

Prepared by Eng. Enas Aldahbali

Student Name			
Department			
Roll #			
Section			
Lab Date	-11-2024		
Submission Date			
Lab Grade:	10	**Obtained Grade**	
Instructor's Signature:			

A. ***Title:*** **Procedural Abstraction and Functions that return a value**

B. ***Objectives of this lab:***

- Learn about Predefined functions
- Learn about Type casting using functions that convert variable types
- Learn about Functions that return a value
- Learn about Procedural abstraction and the importance of argument ordering
- Learn about Scope of variables
- Learn about Function Overloading

Activity 8.1 - Predefined Functions

A function is a subprogram that is included in a C++ program to perform a particular task such as obtaining data, carrying out some calculations, displaying some information or messages, and displaying the output data. In C++, there are many predefined functions that are written to simplify the computations. Following is an example in which a predefined function can significantly simplify the computations. This program is very similar to the one given in the first question in your

Code:-

```
// P31_1.cpp This C++ program computes the value of a to the power of b (a^b) for three cases.
#include<iostream>
using namespace std;
int main(void)
{
    int i = 0, p = 1;
    int a = 2, b = 4;
    while(i < b) // computing 2^4
    {   p = p * a;
        i++;
     }
     cout << a << " to the power of " << b << " is = " << p << endl;
    i = 0;
    p = 1;
    a = 3, b = 3;
    while(i < b)  // computing 3^3
    {    p = p * a;
         i++;
     }
     cout << a << " to the power of " << b << " is = " << p << endl;
    i = 0;
    p = 1;
    a = 5, b = 4;
    while(i < b)  // computing 5^4
    {    p = p * a;
         i++;  }
     cout << a << " to the power of " << b << " is = " << p << endl;
     return 0;
      }
```

prelab The heart of this program is the part in red font. Imagine, you wanted to compute hundred of these calculations in a program. For now, it seems that we have to repeat several lines, which are almost identical, hundred times. But, we may have a better option. Here is a question for you to think about. In program P31_1.cpp, all variables are declared as int. Could you do the same computation for real values, i.e., could you make changes in that program such that it would compute something like $2.3^{5.48}$?
There is a predefined function in C++ that computes a number to the power of another. The function is called *pow*. This function will take two numbers as its **arguments** and will compute one to the power of the other and will return the result which is referred to as **value returned**. For example, in the above program, to compute 3^4, we can use *p = pow(3,4)*. Here, 3 and 4 are the **arguments** to this function and the result, is the **value returned** which is assigned to *p*. Using this predefined function, the above code can be simplified significantly. Please note that in order to use the *pow* predefined function, you need to include the *math.h* directive, i.e. ***#include<math.h>***

Exercise 8-1

Modify the P31_1.cpp program and use the predefined function *pow* to compute the a^b power. Call your new program ex31.cpp. Note that the *pow* function allows us to compute a real number to the power of another real number as well. Here is the definition of the *pow* function: *pow - computes powers* It takes two parameters of type *double*, a and b and its value returned is of type *double* as well, *p = pow(a,b)*
not initialize the choice to 1 this time and compile and run the program. Does the program work the same way?

Code Answer:-

```
1
2
3
4
6
7
8
9
10
11
12
13
```

Exercise 8.2

We can further improve the new program using a *while loop*, so that it asks the user to input an **a** and a **b** (basically of any type) and computes the *pow*(**a, b)** to compute *p = pow(a,b)* and displays the result. Call your new improved program ex32.cpp.

There are more predefined functions in C++, here are some of them:
abs - *computes absolute value* It takes one parameter of type *integer*, a, and its value returned is of type *integer* as well. *p = abs(a)*
fabs - *computes absolute value* It takes one parameter of type *double*, a, and its value returned is of type *double* as well. *p = fabs(a)*
labs - *computes absolute value* It takes one parameter of type *long*, a, and its value returned is of type *long* as well. *p = labs(a)*
sqrt - *computes square root* It takes one parameter of type *double*, a, and its value returned is of type *double* as well. *p = sqrt(a)*
ceil - *ceiling (round up)* It takes one parameter of type *double*, a, and its value returned is of type *double* as well. *p = ceil(a)*
floor - *floor (round down)* It takes one parameter of type *double*, a, and its value returned is of type *double* as well. *p = floor(a)*

As you may remember in Lab2, you were asked to solve the quadratic equation. The solutions to a quadratic equation, $\mathbf{a}x^2 + \mathbf{b}x + \mathbf{c} = 0$, are:

$$x_1 = \frac{-\mathbf{b} + \sqrt{\mathbf{b}^2 - 4\mathbf{ac}}}{2\mathbf{a}}$$

and

$$x_2 = \frac{-\mathbf{b} - \sqrt{\mathbf{b}^2 - 4\mathbf{ac}}}{2\mathbf{a}}$$

The possible solutions were discussed Here, we only mention that before you compute these two roots, you had to make sure that the value under the radical was not negative and that **a** was not zero.

Your Answer:-

```
1
2
3
4
6
7
```

Round up and Round down Functions

Sometimes, you may want to round your numbers up or down. There are two predefined functions that are used for this purpose. The function ceil will round up a given real number to its next whole number and function floor will round down a given real number to its lower whole number.
Examples:

if x = 34.3 then y = *ceil*(x) is 35.0
if x = 34.3 then y = *floor(x)* is 34.0

Exercise 8.3

Answer the following questions:

A) What are the *ceil* and *floor* of 34.6?
B) What are the *ceil* and *floor* of -33.2?
C) Round 34.2, 34.6, -33.2 and -33.7.
D) Suggest a method to round a number using *ceil* or *floor*, then apply it to the numbers in part C.

Answer:-

a)
b)
c)
d)

Absolute Value of Integer and Real Values

In C++ we have two functions that are used to find the absolute value of an integer or a real value. The first one is called *abs* that is used to compute the absolute value of an integer.
Example: x = -4, then y = *abs(x)* will be 4.
The other function is *fabs* that computes the absolute value of a float or double type number.
Example: x = -3.43, then y = *fabs*(x) will be 3.43.

Activity 8-2 - Type Changing Functions (Type Casting)

In problem 3 of the prelab, you were asked to determine a possible source of error in the given program. Perhaps, you have found that dividing two integers will not always result in the correct answer. For example, that program produced 0 when you divide 2 by 4 as two integers. There is a way to fix this problem temporarily, *type casting* that utilizes predefined functions. *Type casting* allows us to change the type of a variable at the line where it is used. For example, in the program given in problem 3, one can change the variable type of the numerator (type cast) to *float*, so that the division will become a float divided by an integer.

Code:-

```
#include<iostream>
using namespace std;
int main(void)
{
    int x,y;
    cout << "Enter 2 values for x and y separated by space, then press <Enter>
:";
    cin >> x >> y;
    cout << endl;
    cout << "With type casting on x " << x << "/" << y << " = " <<
static_cast<float>(x)/y << endl;
    // Just to show you that the type for x is reset back to int again
    cout << "Right after type casting " << x << "/" << y << " = " << x/y << endl;
    return 0;
}
```

The change is shown in red font for clarity. Note that variable x will stay an integer regardless of the static_cast<float>(x)/y statement. Type casting will not change the type for *x*, instead static_cast<float>*(x)* is a function call to the predefined function static_cast<float> with one argument, *x*, with the value returned that is of type *float*. Thus, the division will be the *float* type value of *x* by the *integer* type value *y*, which results in a *float* value.

Exercise 8.4:

The following program is supposed to convert a temperature in degree Fahrenheit to degree Celsius, but it will not produce the correct result. Use type casting to fix the problem and run the program for the test values. Call your new program ex35.C.

Code:-

```
#include<iostream>
using namespace std;
int main( )
{
    int t_in_fah, t_in_cel;  //Notice that we declared these two as integers, not the best choice
    cout << "Enter a temperature in Fahrenheit \n";
    cin >> t_in_fah;

    t_in_cel = 5/9*(t_in_fah - 32);
    cout << "The temperature in Celsius is: " << t_in_cel << endl;

   return 0;
}
```

Test cases:

32 F is 0 C
212 F is 100 C

,

Activity 3-3 - Functions that return a value Procedural Abstraction

We can write our own functions to perform some computations. These type of functions are referred to as **programmer-defined functions**. A programmer-defined function may have three parts: 1) Function definition, 2) Function Call, and/or 3) Function declaration.

The following program, which computes the total cost of purchases made in a store uses a function to compute the cost plus 5% sales tax of purchases made.

Code:-

```
// P33_1.cpp This program computes the total cost of purchases made,
//including 5% sales tax, on number_par items at a cost of price_par each.
#include <iostream>
using namespace std;
double total_cost(int number_par, double price_par); // (1) Function declaration
int main( )
{     double price, bill;
      int number;
      cout << "Enter the number of items purchased: ";
      cin >> number;
      cout << "Enter the price per item $";
      cin >> price;
      bill = total_cost(number, price); // (2) Function call
      // The following three lines are used for formatting purposes.  Since a precision of 2
      //  is set, then all numbers will be displayed with two decimal points.  We work with $
      //  this seems to be the most appropriate way to display the numbers..
      cout.setf(ios::fixed);
      cout.setf(ios::showpoint);
      cout.precision(2);
      cout << number << " items at "
            << "$" << price << " each.\n"
            << "Final bill, including tax, is $" << bill
            << endl;
      return 0;
}
// (3) Function definition
double total_cost(int number_par, double price_par)  // Function heading
{ // Function body begins here
      const double TAX_RATE = 0.05; //5% sales tax, const is to make sure this value stays
unchanged
      double subtotal;
      subtotal = price_par * number_par;
      return (subtotal + subtotal*TAX_RATE);
}// Function body ends here
```

As you can see, we have used a function called *total_cost* to compute the cost + 5% tax. Note that this function has computed the total cost and has returned a single value at return (subtotal + subtotal*TAX_RATE); The returned value is of type *double*.

Remarks

When you work with functions, there are at least 4 things that you must remember .

1) A function must have a name, in the above program the function name was total_cost.
2) A function must have a type, the type for the above function is double.
3) A function must have correct argument definitions. This means that the arguments, if any, must have type consistency and correct ordering at the:
a) function declaration, b) function call, and c) function definition.
4) A function must have the correct return type. In our example, the function was of type double and it returns a value of type double.

Violation of any of these will result in syntax or logical errors.

Procedural Abstraction and Parameter Ordering

A function must be written like a black box. The user does not need to have any knowledge of the details in the body of the function. He/she should be able to determine what the function will do by looking at the function prototype and providing the correct input values with correct types to obtain the correct result from a function.

A function may have several arguments of different types. It is critical that the parameters passed to a function have the same order at a) the function prototype, b) the function call, and c) the function definition. Violation of the correct ordering may result in a) syntax error or b) logical error. It is very hard to find the error when the ordering is incorrect. So if you get an error when you are using a function, the first thing you may want to check is the type and ordering constancy in all instances of the function.

Scope of a variable

The scope of a variable declared inside a function is the body of that function. For example, in the above program variable *bill* isdeclared inside the body of the main function. Thus, *bill* is unknown in the total_cost function. If we attempt to use *bill* without declaring it in the function total_cost, we will get a syntax error that bill is undeclared. Similarly, variable *subtotal* is unknown to the main function, because it is a variable defined in the total_cost function and its scope is the body of the total_cost function only. These types of variables are referred to as **local variables**. They are local to the function in which they are declared. There is, however, a method to define a variable such that it is known to all functions. These types of variables are defined at the top of the program right after the include directives. Any variable defined this way is known as **global variable**.

The parameters passed to a function are also local to the function within which they are defined. Thus, a parameter that may go through changes inside a function that is called by another, will assume its original value upon the completion of the called function. This is referred to as **call_by_value**. The following example will help you understand all these definitions.

Code:-

```
/ P33_2.cpp This program illustrates the local and global variables and call-by-value.
// This program computes the side area and the cross section area of a cylinder
#include<iostream>
#include<cmath>
using namespace std;
double PI = 3.14159;  // This variable is defined globally, known to all functions in this program as PI
double cross_area(double r);  // Function prototype for function cross_area
double side_area(double r, double h);  // Function prototype for function Side_area
int main(void)
{
    double h, r;  //variables local to the main function
     cout << "Enter the radius and the height of the cylinder in Cm <Enter> ";
     cin >> r >> h;
     cout << endl;
     cout << "Before I do any computation or call any function, I want to let you know that \n";
     cout << "you have entered r = " << r << " and h = " << h << "." << endl;
     cout << "I am planning to use inch, thus in the first function, I will convert r, and " << endl;
     cout << "in the second one I will convert h \n";
     cout << "The cross section area of the cylinder is " << cross_area(r) << "  inch-sqr endl;
     cout << "The side area of the cylinder is " << side_area(r,h) << " inch-sqr \n\n";
     return 0;
}
double cross_area(double r)
{
    //Cross secion area includes the disks at the bottom and the top
     r = r * 0.3937;  // converting r to inch
     return 2*PI*pow(r,2);
}
double side_area(double r, double h)
{
     double area; //variable local to Side_area function
     h = h * 0.3937;  // converting h to inch
     area = 2*PI*r*h;
     return area;
```

In the above program, r and h are declared in the main function and are local to that function. These two have been passed to the function side_area with the same names but they are still local to both functions regardless of their names. But, PI is defined globally and will be known to all the functions as PI. If by any chance the value for PI changes in one of these functions, every function which is using PI will use it with its new value. That is why it is a good practice to define a variable that should remain unchanged througout the program as a global variable using a constant modifier, *const.*

Exercise 8.5

Copy or cut and paste program P33_2.cpp to a new program called ex36.cpp. Compile and run the program for the following values:
r = 2 Cm, h = 10 Cm

The answer should be:

The cross section area of the cylinder is 3.8955634 c
The side area of the cylinder is 19.474819 inch-sqr

Did you get the same answer? Explain the reason for such an error and fix the problem.

Answer Code:-

```
1

2

3

4
```

Exercise 8.6

Modify the ex36.cpp to include a new function called total_area, that computes the total suface area of a cylinder. The total surface area is the sum of side area and cross section area. Call your new program ex37.cpp.

For the above test values the total area must be: 23.370382 inch-sqr

Activity 8-4 - Function Overloading

"function name overloading is possible when you have different argument types, different number of arguments, or both"

In the program P33_2.cpp, we used two different function names to distinguish between the function that computes the cross area and the one that computes the side area of a cylinder. Using **overloading** we can give both functions the same name but ask them to do two different things. The decision on which function to be chosen is made based on: 1) difference in the number of arguments, 2) difference between types of parameters, and 3) based on the difference in number and type of parameters (both 1 and 2). Here is the new version of the same program written using overloading.

Code:-

```
// P34_1.cpp This program illustrates the local and global variables and call-by-value.
// This program computes the side area and the cross section area of a cylinder
#include<iostream>
#include<cmath>
using namespace std;
const double PI = 3.14159;  // This variable is defined globally, known to all functions in this program as PI
const double conversion = 0.3937; // This is the Cm to inch conversion factor
double area(double r);  // Function declaration for function that computes cross section area
double area(double r, double h);  // Function declaration for function that computes side area
int main(void)
{
    double h, r;  //variables local to the main function
    cout << "Enter the radius and the height of the cylinder in Cm <Enter> ";
    cin >> r >> h;
    cout << endl;
    cout << "Before I do any computation or call any function, I want to let you know that \n";
    cout << "you have entered r = " << r << " and h = " << h << "." << endl;
    cout << "I am planning to use inch, thus in the first function, I will convert r, and " << endl;
    cout << "in the second one I will convert h \n";
    cout << "The cross section area of the cylinder is " << area(r) << "  inch-sqr endl;
    cout << "The side area of the cylinder is " << area(r,h) << " inch-sqr \n\n";
    return 0;   }
double area(double r)
{  //Cross secion area includes the disks at the bottom and the top
    r = r * conversion;  // converting r to inch
    return 2*PI*pow(r,2);
}
double area(double r, double h)
{   double area; //variable local to Side_area function
    h = h * conversion;  // converting h to inch
    r = r * conversion;  // converting r to inch
    area = 2*PI*r*h;
    return area; }
```

Exercise 8.7

Could we use overloading to compute the surface area and volume of a sphere? Explain your answer. The surface area of a sphere is $S = 4*PI*r^2$ and the volume is $V = (4.0/3.0)*PI*r^3$.

Answer Code:-

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

Note that we were able to use name overloading because of the fact that to compute cross section area, we only needed radius, r, as an argument and to compute the side area, we needed both the radius and height of the cylinder. Thus, here we used the difference between number of parameters to implement name overloading.

Exercise 8.8 :-

Modify program P34_1.cpp to compute the side area, total area, and volume of a cylinder and the area and volume of a sphere, depending on the choice that the user makes. Your program should ask users to enter 1 to choose cylinder or 2 for sphere, and display an "invalid choice error" for other values.

For a cylinder, we want to compute:
- Side area: (2*PI*r) * h
- Total Area: $2*(PI*r^2)$ + Side area
- Volume: $(PI*r^2)*h$

For a sphere, we want to compute:
- Surface area: $4*PI*r^2$
- Volume: $(4.0/3.0)*PI*r^3$.

Use overloading whenever possible.

Call your new program Ex39.cpp.

Code:-

```
1
2
3
4
5
6
7
8
```