CSC10001 – Introduction to Programming

5th lecture: Function

fit hcmus
Instructor: TRAN Thi Thao Nhi

October 31, 2023

Functions

Algebra

$$f(x) = 2x + 5 \quad \Rightarrow f(1) = 7, f(2) = 9, f(3) = 11$$

where

- f: name of function
- $1, 2, 3$: arguments
- $7, 8, 9$: corresponding values of function f

Predefined functions

```
double sqrt(double x)

double result = sqrt(2.25);
```

- function's name: `sqrt`
- parameter[a]: `x`
- argument[b]: 2.25
- return type: `double`

```
double pow(double x, double y)

double result = pow(2.5, 3);
```

- function's name: `pow`
- parameters: `x, y`
- arguments: $2.5, 3$
- return type: `double`

[a] formal parameter/argument
[b] actual parameter/argument

User-defined functions

- *Value-returning functions* return a value of a specific data type using the return statement

```
int abs(int number)
{
    if (number < 0)
    number = -number;
    return number;
}
```

- *Void functions* do not have a return type

```
void displayMessage()
{
    cout << "Hello from the function
    displayMessage.\n";
}
```

Defining and Calling functions

Defining A function definition contains the statements that make up the function

```
functionType functionName(formal parameter list) {
    statements;
}
```

- function's name
- formal parameter list (with datatype)
- return type
- function's body

```
int larger(int a, int b) {
    int max;
    if (a > b) max = a;
    else max = b;
    return max;
}
```

Calling A function call is a statement that causes a function to execute

```
functionName(actual parameter list);
```

- function's name
- actual parameter list (without datatype)

```
larger(20, 25);
int a = 20, b = 25;
int result = larger(a, b);
```

TODO: largest number from 3/n numbers, [DS.Malik] Prog. exercises 8

Flow of execution

- program is compiled sequentially
- `main` function always executes first
- other functions execute only when they are called

```
1 int larger(int a, int b) {
2     ...
3 }
4 int main() {
5     int a = 20, b = 25;
6     int result = larger(a,
      b);
7     return 0;
8 }
```

- compilatation order: (1) → (2) → (4) → (5) → (6) → (7)
- execution order: (4) → (5) → (6) → (1) → (2) → (7)

```
1 int larger(int a, int b);
2 int main() {
3     int a = 20, b = 25;
4     int result = larger(a,
      b);
5     return 0;
6 }
7 int larger(int a, int b) {
8     ...
9 }
```

- compilatation order: (1) → (2) → (3) → (4) → (5) → (7) → (8)
- execution order: (2) → (3) → (4) → (7) → (8) → (5)

Function prototype

A function prototype (function declaration) eliminates the need to place a function definition before all calls to the function

```
functionType functionName(parameter list);
// function heading without the body
```

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

// function prototypes
int larger(int a, int b);
int largest(int, int, int);

int main() {
    int a = 20, b = 30, c = 25;
    int result = largest(a, b, c);
    cout << "The largest is " << result;
    return 0;
}
// function definition
int largest(int a, int b, int c) {
    int max = larger(a, b);
    ...
}
int larger(int a, int b){
    ...
}
```

Returning

The `return` statement causes a function to end immediately

```
int larger(int a, int b) {
    if (a > b) return a;
    return b;
}
float sum(float a, float b)
    {
    return a + b;
}
```

```
void printLarger(int a, int b) {
    if (a > b) {
        cout << a << " is larger";
        return;
    }
    cout << b << " is larger";
}
```

A value-returning function will use `int`, `double`, `bool`, or any other valid data type in its header

Some peculiarity:

```
int larger(int a, int b) {
    if (a > b) return a;
    // meet a warning
}
```

```
int larger(int a, int b) {
    if (a > b) return a;
    return b, a; // logical
     error
}
```

Function parameter

Default arguments are passed to parameters automatically if no argument is provided in the function call

- must be the far-right when defining and calling
- must be a literal value or a named constant
- can be changed if necessary
- can listed in the function prototype or function heading
- must be occurred before `main function`

```
void func(int a, double d, char c = 'A', int u = 25);
// calling
func();
func(a, d);
func(10, 32.6, 'g', 12);
//errors
void func(int a, double d, char c = 'A', int u);
void func(int a, double d = 1.2, char c, int u = 5);
func(10, 'x', 10);
func(10, 'x');
```

Function parameter

Passing data by Value When an argument is passed into a parameter, only a copy of the argument's value is passed. Changes to the parameter do not affect the original argument

```
void swap(int a, int b) {
    int temp = a; // save the original a
    a = b; // update a
    b = temp; // update b
}
```

Passing data by Reference When used as parameters, reference variables allow a function to access the parameter's original argument. Changes to the parameter are also made to the argument

```
void swap(int &, int &); //prototype

//ampersand & in front of variable name
void swap(int &a, int &b) {
    int temp = a; // save the original a
    a = b; // update a
    b = temp; // update b
}
```

TODO: [DS.Malik] Prog.exercises 7

- func: input day, month, year1
- func: check if leap year
- func: calculate the day number (normal year)
- func: calcualate the day number (leap year)

Variables

The scope of an identifier refers to where in the program an identifier is accessible (visible)

A local variable is defined inside a function and is not accessible outside the function

A global variable is defined outside all functions and is accessible to all functions in its scope

```
1  int g = 5; // g: global variable
2  void func(int , int);
3  int main() {
4      int x = 3, t = 10; //x, t: local variables
5      cout << g << " ";
6      func(x, t); cout << g << " ";
7      func(g, t); cout << g << " ";
8      return 0;
9  }
10 void func(int a, int threshold) {
11     g = a;
12     a = a > threshold ? -a : 3*a;
13     //a, threshold: local variable
14 }
```

Variables

An automatic variable is variable for which memory is allocated at block entry and deallocated at block exit

```
void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
=> a, b, temp are automatic variables
```

A static variable is variable for which memory remains allocated as long as the program executes

- global variables are static variables
- self-defined static variable:

```
static dataType identifier;
```

- Static variables declared within a block are local to the block, and their scope is the same as that of any other local identifier of that block

Function overloading

Two or more functions may have the same name, as long as their parameter lists are different

- number of formal parameters
- datatype in at least one position

```
void func(int a);
void func(int a, int b);
void func(double a, int b);

int func(int a); // error
```

TODO: calculate the perimeter of some shapes

- square, 1 input, $4a$
- rectangle, 2 input, $2(a + b)$
- triangle, 3 input, $a + b + c$

TODO

- Finish chapter 6
- Read chapter 7.1-7.8;
  + [DS.Malik] chapter 9 part Arrays