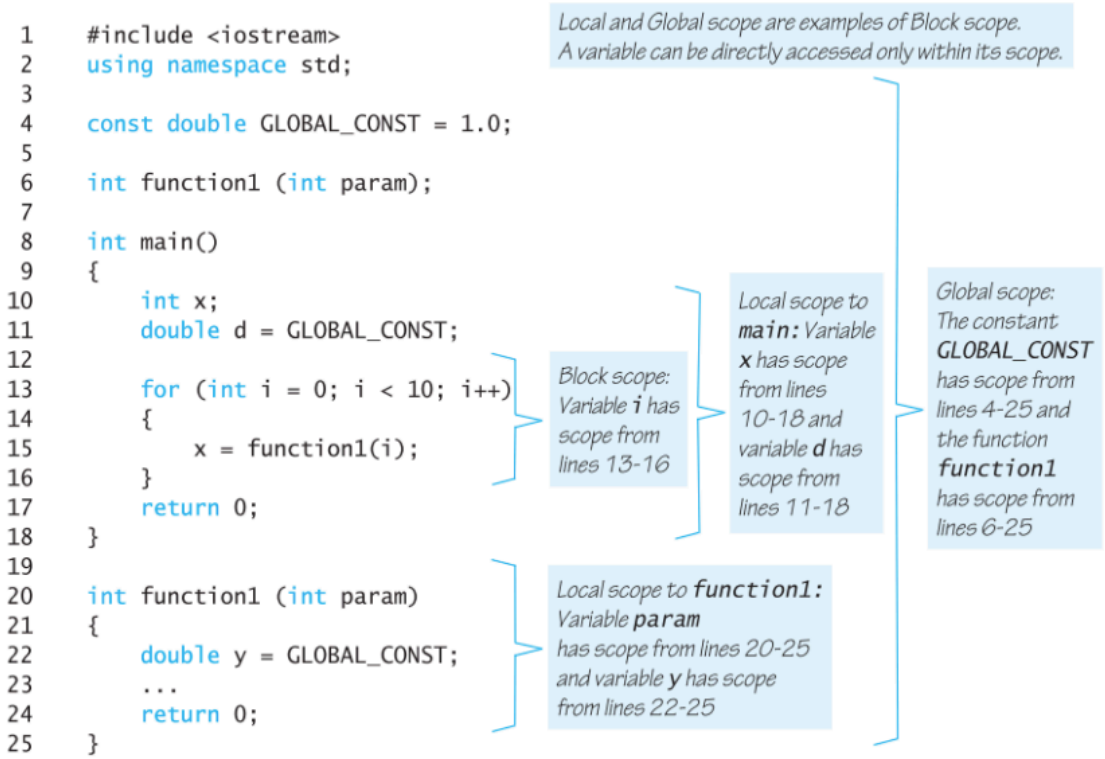
In-class Worksheet #9
Fall 2022 ， Nov 11, Nov 15, 2022
Computer Science I & Lab

1. Scope: Any identifier used in a C++ program (such as the name of a variable or object, the name of a type or class, or the name of a named constant) has a *scope*, i.e., a region of the program in which that name can be used. Often we are mostly concerned with the scope of variables.

   A variable's scope is decided by where it's defined/declared.




```
1   #include <iostream>
2   using namespace std;
3
4   const double GLOBAL_CONST = 1.0;
5
6   int function1 (int param);
7
8   int main()
9   {
10      int x;
11      double d = GLOBAL_CONST;
12
13      for (int i = 0; i < 10; i++)
14      {
15          x = function1(i);
16      }
17      return 0;
18  }
19
20  int function1 (int param)
21  {
22      double y = GLOBAL_CONST;
23      ...
24      return 0;
25  }
```

Some points of discussions:

a) Can we access variable x in function1? Can we access variable y in main?

No, we cannot access x from function1, as x is local to main function. Similarly, we cannot access y, the local variable in function1 from main.

b) Can we have two local variables in function1 with same name?

No, as the two variables will have same scope, and we won't be able to differentiate them by name.

c) Can we define another variable named x in function1?

Yes. We can, this local variable in function1 will have different scope from the variable x in main. In another words, x in function1 will refer to this local variable of x; and reference to x in main() will refer to main's local variable x.

d) Can we define another variable named x inside the for loop's body, line 15?

Actually, we can. This variable x (defined in the body of for loop) has a block scope (line 14-16, i.e., within the curly braces). This means if we write x inside this curly brace, it will refer to this variable x. If we refer to x anywhere else in main() function, the it refers to the main()'s local variable.

We say that within the curly braces, the main's variable x is ***eclipsed by*** *the block local variable x (i.e., it cannot be accessed by x anymore).*

e) Can we declare a local variable named param inside function1?

No. As function1 already has a parameter (which is a special kind of local variable) named param. We won't be able to differentiate these two variables with same name and same scope.

Function formal parameters are actually local variables of the function, and they are initialized (i.e., assigned values) at the function call.

f) Suppose we want to modify x, the local variable in main within function1?

This is possible, but we need to use a new way of passing parameters: pass-by-reference parameter.

2. Function Design Principle (KISS: Keep It Simple and Stupid)

Each function should be designed to do one thing (i.e., has a well-defined functionality) well. For example, a function can be designed to check if a number is even or not, and it should only return the checking result (true or false). We should not use cout statement to output the checking result, or read in the number from cin…

Functions designed following this principle is versatile (i.e., can be used in many different settings/applications/programs).
A related principle is that we want to design the interface of the function (i.e., the parameters and return type) to be as simple as possible.

3. Four common types of functions with examples

a) Functions that perform some calculation or conversion, and return the result

A black-box analogy: something comes into the box and out comes the result of the calculation/conversion.

Input will be the parameters; output will be the return value.

Requirement: Write a function that calculates the average of three integer values

Step 1: Analysis: Input to the function is the three integers, output is their average (a double value):

Step 2: declare the function, i.e., figure out the name, parameters and return types…

Step 3: Implement the function…

```
// Calculate and return average of three integers
double average (int a, int b, int c)
{
        //Implementing this function, independently of other codes,
        //focusing on what you need to output as return value
        int sum;        //use as many local variables as needed, just like in main…
        sum = a+b+c;

        return (sum/3.0);
}
```

b) *Predicate Functions:* Functions that check if some certain conditions are true or not. Such function should return a bool type value.

For example, it's very common that we need to check whether user input is valid or not. We can write a function to implement such functionality to avoid repeat same code when reading multiple inputs. It's a good idea to name same function with Is…, to convey that it answers a true/false question for the input.

```
//Check if a date is valid or not
bool IsValidDate (int year, int month, int day)
{
    if (year<0 || month >13 || month<1)
        return false;

    if (day < 1 || day > DaysInMonth (year, month))
        return false;
```

```
    //We have checked for all conditions for the date to be invalid
    return true;
}
```

c) "Output Functions": Functions that output/display data in the terminal (or other output devices, such as files)

Such functions usually do not return value, instead they output the data/results to terminal, as a result, denote the return type as "void" to indicate that there is no return value.

e.g., write a function that prints a divider line making up of 30 * to the terminal. How to make it so that you can tell it to print using different character, with different length?

```
// display a divider line in terminal using the symbol given, and of the given
length
void PrintDividerLine (char symbol, int length)
{
     //when implementing the function, only do what's expected…
     for (int i=0; i< length; i++)
         cout <<symbol;
    cout <<endl;
}
```

Write a function to display a time (hr, min, sec) to terminal in the format of HH:MM::SS?

```
//Ouput a time in the format of HH:MM:SS
void PrintTime (int hr, int min, int sec)
{
    cout <<setw(2) << setfill('0') << hr <<':'
          <<setw(2) << setfill ('0') <<min<<':'
         <<setw(2) <<setfill('0')<<sec;
   //note that we do not output endl here
}
```

When calling this function from main (e.g. in lab5):

```
 If (time1 is same as time2) {
     PrintTime (hr1, min1, sec1);
     cout <<" is same as ";
     PrintTime (hr2, min2, sec2);
     cout<<endl;
 }
```

d) "Input Functions": Functions that reads data from keyboard (or other input devices, such as files)

For example, in lab1, we want to read the number of pizza ordered. This can be built as a function if we want to reuse the code to read different types of pizza, and check if the input is valid or not, and potentially keep trying until a valid input is entered.

Such functions might not have any parameter:

```
//Read an non-negative number, and keep on trying until a non-negative value is entered
int ReadNonNegativeNumber ( )
{
      int num;

      cout <<"Enter a non-negative number:";
      cin >> num;

       while (num<0)
       {
           cout <<"Input is negative. Try again:";
           cin >> num;
        }
       return num;
}
```

Or it might take some parameter to help customize the prompt message:

```
//Read a pizza order
int ReadNumberOfPizzaOrdered (string type )
{
      int num;

      cout <<"How many " << type <<" pizzas to order:";
      cin >> num;

       while (num<0)
       {
           cout <<"Input is negative. Try again:";
           cin >> num;
        }
       return num;
}
```

What if we want to reads multiple variables which make up the data (e.g., each date is expressed by three integers: year, month and day)?

The following naïve way to do it does not work. Do you know why?

```
void ReadDate (int year, int month, int day)
{
     char ch;
       cin >> year >> ch >> month >> ch >> day;
       while (!IsValidDate (year, month, day)) {
            cout << "Invalid input. Try again (YY/MM/DD):";
            cin >> year >> ch >> month >> ch >> day;
       }
       // with or without return;
}

 int main()
{
    Int year, month, day;
    Cout <<"Enter your birthday:";
    ReadDate (year, month, day);
    Cout <<"you entered:" << year <<"/"<<month<<"/"<<day<<endl;

}
```

- We will learn struct type which allows use to group multiple variables into one compound type variable later.
- For now, we can use **pass-by-reference parameters**

4. Pass-by-reference parameters

Pass-by-value parameters: So far, all examples we have seen use pass-by-value parameters: the value of the arguments are assigned to the parameters at the function call.

The argument for such parameter can be anything that has a value of the given type, e.g., sqrt(11.23), sqrt (a); sqrt (sqrt(a)), ….

Pass-by-argument parameters: at the function call, the addresses of the arguments (which must be variables) are assigned to the parameters at the function call, essentially, this means "using this variable as this parameter"…

Syntax: to denote that a parameter should be "passed-by-reference", add a & between the type and name of the parameter:

```
//A correct way to read a date: using three pass-by-reference parameters
void ReadDate (int & year, int & month, int & day)
{
      char ch;
      cin >> year >> ch >> month >> ch >> day;
      while (!IsValidDate (year, month, day)) {
           cout << "Invalid input. Try again (YY/MM/DD):";
           cin >> year >> ch >> month >> ch >> day;
      }
      // with or without return;
}

int main()
{
   int tyear, tmonth, tday;
   cout <<"Enter today's date:";

   // use main's tyear, tmonth, tday local variable as year, month, day
   ReadDate (tyear, tmonth, tday);
   cout <<"you entered:" << tyear <<"/"<<tmonth<<"/"<<tday<<endl;

   int byear, bmonth, bday;
   cout <<"Enter your birthday:";

  // use main's byear, bmonth, bday local variable as year, month, day
   ReadDate (byear, bmonth, bday);
   cout <<"you entered:" << year <<"/"<<month<<"/"<<day<<endl;

}
```

Details: how to trace function call when there is pass-by-reference parameter…

- All variables are stored in main memory. The main memory consists of a long list of numbered locations called memory locations. The number associated with a byte is called its address.
- A group of consecutive bytes is used to store a data item, and the address of the first byte in the group is used as the address of the data item.
- Recall: a char takes one byte, an int is 4 bytes, …
- sizeof() function can be used to obtain the size of a data type

In the function call, the actual arguments' addresses are assigned to *the parameters, and year (month, day) becomes a nickname for the local variables tyear (tmonth, tday)* during the function call:

ReadDate (tyear, tmonth, tday);

i.e., the year, month, day inside ReadDate() function refers to the tyear, tmonth, tday….

More later: you can use reference variable in other situation:

```
int a=10;
int & b=a; //b is a nickname for variable a, i.e., one variable with two names!
b=20;
```