SWE3053: Basic and Practice in Programming

- Lab 4: Function

In this lab ...

- Function
- Scope of Variables

- What you need to submit in this lab (Lab #4):
  - Lab Exercise #4 by Wednesday 11:59 pm
  - Lab Assignment #4 by Tuesday 11:59 pm

Function

1. Code Reuse
   - So that you don’t have to write the same code again and again
   - Write a function for common task

2. Modularization
   - Organize your code – break long code into chunks

To write a function

1. Function Prototype
2. Function Call
3. Function Definition

```
#include <stdio.h>

int MyFunction(int input);

int main(void)
{  …
   int x = 89;
   int result = MyFunction(x);
   …
}

int MyFunction(int input);
{  int output = input - 10;
   return output;
}
```

1. Function Prototype

- For the compiler: used for validating function calls
- Function Prototype is not necessary if you put function definition before the function call

```
#include <stdio.h>

int MyFunction(int input);

int main(void)
{  …
   int x = 89;
   int result = MyFunction(x);
   …
}

int MyFunction(int input);
{  int output = input - 10;
   return output;
}
```

Function Prototype

- Both are valid

```
#include <stdio.h>

int MyFunction(int input);

int main(void)
{  …
   int x = 89;
   int result = MyFunction(x);
   …
}

int MyFunction(int input);
{  int output = input - 10;
   return output;
}
```

```
#include <stdio.h>

int MyFunction(int input);
{  int output = input - 10;
   return output;
}

int main(void)
{  …
   int x = 89;
   int result = MyFunction(x);
   …
}
```

Function Definition

Define your function:

return-value-type function-name(*parameter-list*)
{
    definitions
    statements
}

For example:

Return value is `int`   Function name is `MyFunction`   Function takes one input (`int`)

```
int MyFunction(int input)
{ int output = input - 10;
  return output;
}
```

Function Definition

- Your function can have no return value

```
void MyFunction(int input)
```

- Your function can take no input

```
int MyFunction(void)
```

- Your function can take more than one input

```
char MyFunction(int input1, float input2, char input 3)
  { int result = input1 + input2;

    if(result>=60)  return ‘p’;
    else            return ‘f’;
  }
```

Why Writing Functions?

- Code reuse
  - So that you don’t have to write the same code again
- Modularization
  - To organize your code

Lab Exercise 4

- You are giving this following code segment
- You need you complete the function definition of the 3 functions
- Your program will ask user to input an integer, and then calculate and display its cubic value
- You program should behave as in the following page

```
#include <stdio.h>
#include <math.h>

void PrintMenu(void);
int AskUserInput(void);
void DisplayResult(int result);

int main(void)
{ PrintMenu();
  int input = AskUserInput();
  int result = pow(input,3);
  DisplayResult(result);
  return 0;
}

void PrintMenu(void)
{
}

int AskUserInput(void)
{
}

void DisplayResult(int result)
{
}
```

Complete these 3 functions

Lab Exercise 4 Sample Outputs

```
Please input an integer: 10
You entered: 10
The result is: 1000

--------------------------------
Process exited after 1.187 seconds with return value 0
Press any key to continue . . .
```

```
Please input an integer: 3
You entered: 3
The result is: 27

--------------------------------
Process exited after 1.187 seconds with return value 0
Press any key to continue . . .
```

Scope of Variables

- When you declare a variable, that name and value is only "alive" for some parts of the program
- What is a variable's scope?
  - Starts at the declaration statement
  - Ends at the end of the block it was declared in
- If the variable is declared within a block (compound statement, { }) it only stays alive until the end of the block
- Applicable to functions, `if` statement, loops (`for`, `while`, `do ... while`), etc

Lifetime of Variables

- A variable is created when its scope is entered during the execution of a program
- A variable is destroyed when its scope is left during the execution of a program

```
{ // enter scope here
    int local_x;
    // some statements using local_x
 } // leave scope here
```

- …When the program enters the scope local_x is created
- …When the program leaves the scope local_x is destroyed
- …When the program enters the scope again a NEW variable called local_x is created
- The lifetime of a global variable is the same as the lifetime of the program
- …A global scope is created when a program is started and destroyed when the program finishes

Scope of Variables: Example

- Example:

```
for(int x; x<=10; x++)
{ printf("X: %d\n",x);
}
```

Variable x is declared here

Scope of variable x

- Another example:

```
int main(void)
{ int x;
  …
  return 0;
}
```

Variable x is declared here

Scope of variable x

```
int FunctionA(void)
{ …
  return 1;
}
```

Outside of the scope.
Variable x cannot be referenced

Scope of Variables: Example

- Example:

```
for(int x; x<=10; x++)
{ printf("X: %d\n",x);
}
printf("X: %d\n",x);   //ERROR! Referencing x outside scope
```

Variable x is declared here

Scope of variable x

- Another example:

```
int main(void)
{ int x;
  …
  return 0;
}

int FunctionA(void)
{ …
  printf("X: %d\n",x);   //ERROR! Referencing x outside scope
  return 1;
}
```

Variable x is declared here

Scope of variable x

Outside of the scope.
Variable x cannot be referenced

Scope of Variables: Example

- Another example:

```
int main(void)
{ int x;
  …
  return 0;
}
```

Variable x is declared here

Scope of variable x

```
int FunctionA(void)
{ int x;
  …
  return 1;
}
```

Another variable x is declared here

Scope of the second variable x

- The two variables `x` are two different variables (with the same name but different scope)
- As a good programmer, you should avoid this (this causes confusion!).

Global Variables

- Example:

```
#include <stdio.h>
void FunctionA(void);
int x = 0;

int main(void)
{ printf("X: %d\n",x);  // X: 0
  x = 10;
  printf("X: %d\n",x);  // X: 10
  FunctionA();
  printf("X: %d\n",x);  // X: 25
  return 0;
}

void FunctionA(void)
{ printf("X: %d\n",x);  // X: 10
  x = 25;
}
```

Static Variables

- Example:

```
#include <stdio.h>

int MyFunction(int input);

int main(void)
{ MyFunction();  // X: 4
  MyFunction();  // X: 5
  MyFunction();  // X: 6
  MyFunction();  // X: 7
}

void MyFunction(void)
{ static int x = 3
  x = x + 1;
  printf(“X: %d\n”,x);
}
```

Static Variables

- When the variable is declared as static, it exists during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope.
- At the end of the scope, static variable is not destroyed and its value is retained.
- Therefore, making local variables static allows them to maintain their values between function calls.

```
#include <stdio.h>
int MyFunction(int input);

int main(void)
{ MyFunction();  // X: 4
  MyFunction();  // X: 5
  MyFunction();  // X: 6
}

void MyFunction(void)
{ static int x = 3
  x = x + 1;
  printf(“X: %d\n”,x);
}
```

Lab Assignment #4: Function

Rewrite the program you wrote in **Lab Assignment #3**

- Your program will function exactly the same.
- **However, you will rewrite the currency conversion using a function:**

```
float CurrencyConversion(float AmountInWon, float rate)
```

- This function will convert any amount in Korean Won into a foreign currency based on an exchange rate

Lab Assignment #4: Function

Sample output 1:

```
Please choose which currency you want to convert:
A - Korean Won to US Dollar (Exchange Rate: 0.000905)
B - Korean Won to Euro (Exchange Rate: 0.000807350908)
C - Korean Won to Japanese Yen (Exchange Rate: 0.0919061643)
D - Korean Won to Chinese RMB (Exchange Rate: 0.00603703605)
E - Quit
Enter your option: A
Enter the amount in Korean Won: 10000
10000 Won equals to 9.050000 USD

Please choose which currency you want to convert:
A - Korean Won to US Dollar (Exchange Rate: 0.000905)
B - Korean Won to Euro (Exchange Rate: 0.000807350908)
C - Korean Won to Japanese Yen (Exchange Rate: 0.0919061643)
D - Korean Won to Chinese RMB (Exchange Rate: 0.00603703605)
E - Quit
Enter your option: B
Enter the amount in Korean Won: 10000
10000 Won equals to 8.073509 Euro

Please choose which currency you want to convert:
A - Korean Won to US Dollar (Exchange Rate: 0.000905)
B - Korean Won to Euro (Exchange Rate: 0.000807350908)
C - Korean Won to Japanese Yen (Exchange Rate: 0.0919061643)
D - Korean Won to Chinese RMB (Exchange Rate: 0.00603703605)
E - Quit
Enter your option: E
```

Lab Assignment #4: Function

Sample output 2:

```
Please choose which currency you want to convert:
A - Korean Won to US Dollar (Exchange Rate: 0.000905)
B - Korean Won to Euro (Exchange Rate: 0.000807350908)
C - Korean Won to Japanese Yen (Exchange Rate: 0.0919061643)
D - Korean Won to Chinese RMB (Exchange Rate: 0.00603703605)
E - Quit
Enter your option: F
You entered an invalid input.

Please choose which currency you want to convert:
A - Korean Won to US Dollar (Exchange Rate: 0.000905)
B - Korean Won to Euro (Exchange Rate: 0.000807350908)
C - Korean Won to Japanese Yen (Exchange Rate: 0.0919061643)
D - Korean Won to Chinese RMB (Exchange Rate: 0.00603703605)
E - Quit
Enter your option: C
Enter the amount in Korean Won: 10000
10000 Won equals to 919.061646 Yen

Please choose which currency you want to convert:
A - Korean Won to US Dollar (Exchange Rate: 0.000905)
B - Korean Won to Euro (Exchange Rate: 0.000807350908)
C - Korean Won to Japanese Yen (Exchange Rate: 0.0919061643)
D - Korean Won to Chinese RMB (Exchange Rate: 0.00603703605)
E - Quit
Enter your option: E
```

Lab Assignment #4: Function

Submit your source code on iCampus before Tuesday 11:59 pm