DATA STRUCTURE & PROGRAMMING II

Topic: Recursive function

Outline

- Review on Function

- What is recursive function?

- How to use recursive

- Examples and Practices

Review on Function

Function

What and Why?

- A function is a block of code that performs a specific task. A function may return a value.

- Why function?
  - ✓ A large and complex program can be divided into smaller programs
    - ✓ assign task and work in team is easy
  - ✓ Program is easy to understand, fix error, and maintain
  - ✓ Reusable code
  - ✓ A function can be call again and again anytime and anywhere in the program

Function

❑ Type of function

- There are two types of functions in C programming (likewise, in C++):

  1. Standard library functions

     - Also called: built-in function, or existing function

     - For example, see Table 1

  2. User-defined functions

     - Also called: user-custom function

     - For example, a user (programmer) defines his/her own function to do something

Library	Provided functions
`stdio.h`	`printf(), scanf()`
`math.h`	`pow(), sqrt(), ceil(), floor(), cos(), sin()`

Syntax C language

c) Parameter of a function
It has parameter type and parameter name
If have more than one, separate them by comma

```c
type functionName(type parameter1){
    ... .. ...
    ... .. ...
    return value;
}
```

b) Name of function
it should start with a verb

a) Type of value returning
from the function

d) Return value
It must be the same data type as a)

```c
int sumTwoNumber(int n1, int n2){
    int s;
    s=n1+n2;
    return s;
}
```

Ex1: Create a function to do
summation of two numbers

6

Syntax C language

❑ Create Function with no returning value

```
void functionName(type parameter1){
    ... .. ...
    ... .. ...
    return value;
}
```

c) Parameter of a function
It has parameter type and parameter name
If have more than one, separate them by comma

b) Name of function
it should start with a verb

a) Void means no returning value from the function

d) No need to *return value* since we use *void* in a)

```
void greetMessage(char name[20]){
    printf("Hi, %s", name);
    printf("Welcome back!");
}
```

Ex1: Create a function to display a greet message for a person

Example of functions

- Function with returning value
- Function with no returning value

```
int sum(int a, int b){
    int res=0;

    res=(a+b)*2;
    n++;
    return res;
}
```

```
void display(int a){
    printf("Hello %d\n", a);
    printf("This function has no
returning value");
    printf("use the keyword void");
}
```

Syntax C language
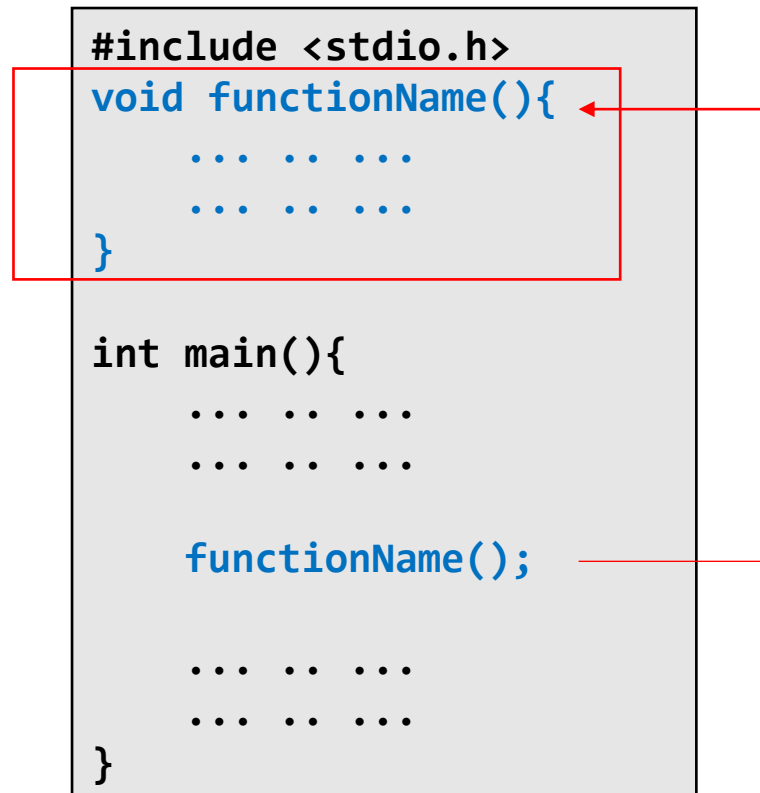
❑ How to execute the created function?

▪ The execution of a C program begins from the **main( )** function.

▪ Our user-defined function can be executed by calling it from **main()** function

```
#include <stdio.h>
void functionName(){

    ... .. ...

    ... .. ...
}


int main(){

    ... .. ...

    ... .. ...


    functionName();



    ... .. ...

    ... .. ...

}
```
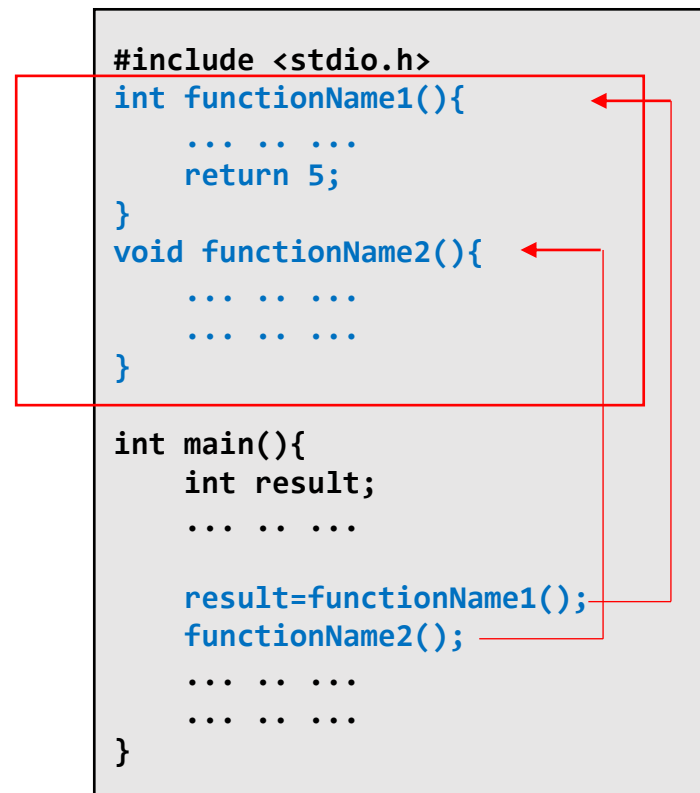
```
#include <stdio.h>
int functionName1(){

    ... .. ...
    return 5;
}
void functionName2(){

    ... .. ...

    ... .. ...
}

int main(){
    int result;
    ... .. ...


    result=functionName1();
    functionName2();

    ... .. ...
    ... .. ...
}
```

Function

❑ Global Vs. Local variable

- **Local variable** a variable that creates inside a function
  - ✓ Its value can not be accessed from outside the function
  - ✓ Its value will be destroyed after the function finish executing

- **Global value** is variable that creates

outside the function below all library inclusion
  - ✓ Its value can be accessed from any functions
  - ✓ Its value will be destroyed after the whole program finish executing

Global variables

User-defined Function

Local variables

Local variables

Call created function

Output:

```
*Start program
14
14
2
```

```c
#include <stdio.h>

int n=0;
char name[10];
int sum(int a, int b){
    int res=0;

    res=(a+b)*2;
    n++;
    return res;
}


int main(){
    int num=10;
    int res=2;

    printf("*Start program \n");
    printf("%d", sum(2,5));
    res = sum(2,5)
    printf("%d", res);
    printf("%d", n);


}
```
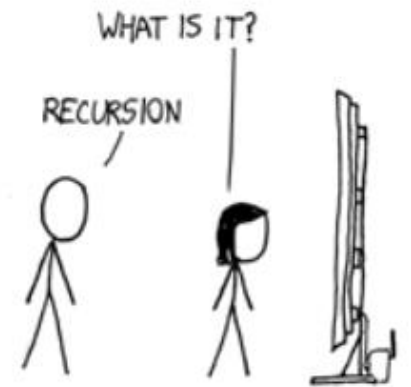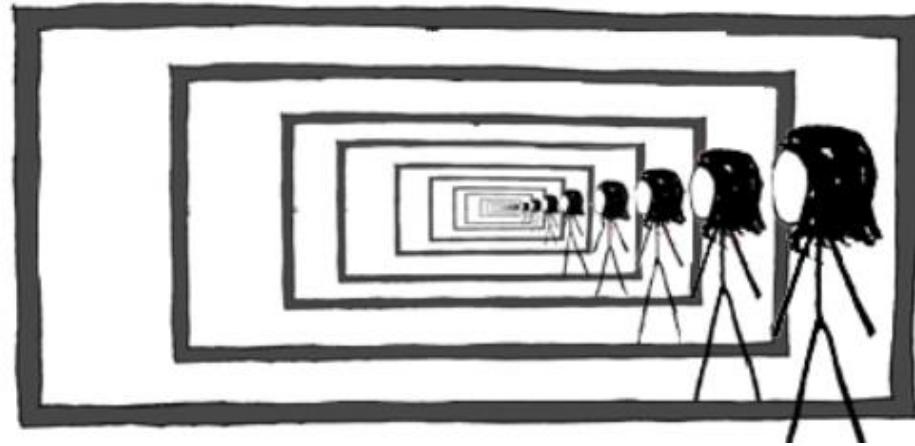
Recursive Function

Recursive

❑ Definition

- **A recursion** is an algorithm which calls itself with smaller (or simpler) input values

- In programming languages, *recursion* is the ability given to a function to call itself

- To write a recursive program, the following steps are needed

  1. Find a recursive **decomposition of the problem** (find cases which allow to call itself again)

     - Express solution in general case

  2. Find the stop condition (find base case)

     - Check that the stop condition is reached after a finite number of recursive calls in all cases

Recursive

❑ Definition

- **Recursive function** is a function that calls itself repetitively until certain condition is satisfied. It works like a loop

- Component of recursive function

  - A condition to determine when the function should **call itself again** (**keep repetition**)

    [repeat case or general case]

  - A condition to determine when the function should **not call itself again** (**stopping condition**) [base case]

Remark: When call itself again, we should update its parameter in order to reach the stopping condition

  - **If-else** statement

  - A *return* statement

Recursive

❑ Example of recursion in factorial

▪ The factorial of a positive integer n, denoted by

   ▪ n! = 1*2*…*(n-1)*n = (n-1)! *n

▪ To calculate **n!**, we just know how to calculate **(n-1)!** then multiple by **n**

▪ The subproblem **(n-1)!** is the same as the initial problem, BUT a simpler case since **n-1 < n**

How to do it in coding?

Recursive

- Calculate factorial

  - n! = 1   if n<=1

  - n! = ((n-1)!) * n   if n>1

How does it work

✓ fac(4)

  ✓ retrun fac(3)*4=24

    ✓ return fac(2)*3=6

      ✓ return fac(1)*2=2

        ✓ return 1

```
Function fac(n: integer): integer
Begin function
        if(n<=1) then
                return 1
        else
                return (fac(n-1))*n
        end if
End function
```

An example of a recursive function to calculate a factorial number

16

Example of recursion in Suit Fibonacci

0 1 1 2 3 5 8 13 21 34

Recursive

❑ Example of recursion in Suit Fibonacci

- Suit Fibonacci

  - $U_n = 1$ if n<=2

  - $U_n = U_{n-1} + U_{n-2}$ if n>=3

```
Function fibo(n: integer): integer
Begin function
        if(n==1 OR n==2) then
                return 1
        else
                return fibo(n-1)+fibo(n-2)
        end if
End function
```

An example of a recursive function in pseudo code to calculate the nth element of suit fibonacci

Recursive

- Sum

  - F(n) = n + f(n-1)   when n>1

  - F(n)=1   when n=1

```
Function sum(n: integer): integer
Begin function
        if(??) then
                return ??
        else
                return ??
        end if
End function
```

An example of a recursive function in pseudo code to computer summation

Using recursive to find summation
S = 1 + 2 + 3 + … + n

Recursive

❑ Remark

- **Base case** is a case that it does not call to the main problem. It stops recursive

- In each call of recursive, it should tend to reach the base case.

Recursive: Direct and Indirect

❑ Definition

- Direct recursive

  - Subproblem uses the main problem to call for defining the result

- Indirect recursive

  - Subproblem A calls subproblem B

- Example

```
f(x)=1                  if x=1 or x=2
f(x)=f(x-1)*f(x-2)      if x>2
```

```
f(x)=1                  if x=1
f(x)=g(x)+2             if x>1
g(x)=0                  if x=2
g(x)=f(x)-1             if x>2
```

Direct recursive   Indirect recursive

Practice

Write a C++ program using recursive to ….

1. Calculate the summation $(2 + 4 + 6 +… + n)/n$ using recursive where n is a number input by a user

2. Display the message Hello 1, Hello 2, …, Hello n, where n is an input given from user.

3. Count the number of digits in an integer input by a user using recursive function.

Q & A