Function

Lecture 06

Min Zhang

zhangmin@sei.ecnu.edu.cn

2020.11.02

Software Engineering Institute

10:00-11:40, Monday, Room 319

Software Engineering Institute, East China Normal University

Review: Summary of Chapter 3

Control flows:

1. `if`-statement
2. `switch`-statement
3. `while`-statement
4. `for`-statement
5. `do-while`-statement
6. `break`
7. `goto`-statement

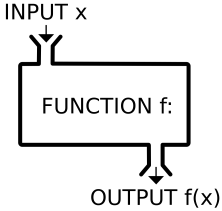
☞ **function (函数)**

Function in mathematics

$$f\colon X \to Y \text{ s.t. } f(x) = 2x + 1$$

Function in mathematics

$$f : X \to Y \text{ s.t. } f(x) = 2x + 1$$



Function in C language

```c
int f(int x){      // f: ℤ → ℤ
  int y = 2*x+1;   // f(x) = 2x + 1
  return y;
}

int a=f(3);        // a will be 7
```

Some functions you have used

- `main()`

Some functions you have used

- `main()`
- `scanf("...",...)`

Some functions you have used

- `main()`
- `scanf("...",...)`
- `printf("...",...)`

Some functions you have used

- `main()`
- `scanf("...",...)`
- `printf("...",...)`
- `getchar()`

Some functions you have used

- `main()`
- `scanf("...",...)`
- `printf("...",...)`
- `getchar()`
- `gets(s)`

Some functions you have used

- `main()`
- `scanf("...",...)`
- `printf("...",...)`
- `getchar()`
- `gets(s)`
- ...

Some functions you have used

- `main()`
- `scanf("...",...)`
- `printf("...",...)`
- `getchar()`
- `gets(s)`
- ...

Some functions you have used

- `main()`
- `scanf("...",...)`
- `printf("...",...)`
- `getchar()`
- `gets(s)`
- ...

The commons of them:

- they are not reserved words

Some functions you have used

- `main()`
- `scanf("...",...)`
- `printf("...",...)`
- `getchar()`
- `gets(s)`
- ...

The commons of them:

- they are not reserved words
- followed by parentheses

Some functions you have used

- `main()`
- `scanf("...",...)`
- `printf("...",...)`
- `getchar()`
- `gets(s)`
- ...

The commons of them:

- they are not reserved words
- followed by parentheses
- there may be expressions in the parentheses

Some functions you have used

- `main()`
- `scanf("...",...)`
- `printf("...",...)`
- `getchar()`
- `gets(s)`
- ...

The commons of them:

- they are not reserved words
- followed by parentheses
- there may be expressions in the parentheses
- they have values (like expressions), e.g., `scanf("%c",&c)!=EOF`

What is a function?

About function from Wiki

In computer programming, a subroutine is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

In different programming languages, a subroutine may be called a procedure, a function, a routine, a method, or a subprogram.

What is a function?

About function from Wiki

In computer programming, a subroutine is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

In different programming languages, a subroutine may be called a procedure, a function, a routine, a method, or a subprogram.

Example (Examples of specific tasks)

1. output data

What is a function?

About function from Wiki

In computer programming, a subroutine is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

In different programming languages, a subroutine may be called a procedure, a function, a routine, a method, or a subprogram.

Example (Examples of specific tasks)

1. output data

What is a function?

About function from Wiki

In computer programming, a subroutine is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

In different programming languages, a subroutine may be called a procedure, a function, a routine, a method, or a subprogram.

Example (Examples of specific tasks)

1. output data — e.g., `printf`
2. read data

What is a function?

About function from Wiki

In computer programming, a subroutine is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

In different programming languages, a subroutine may be called a procedure, a function, a routine, a method, or a subprogram.

Example (Examples of specific tasks)

1. output data — e.g., `printf`
2. read data

What is a function?

About function from Wiki

In computer programming, a subroutine is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

In different programming languages, a subroutine may be called a procedure, a function, a routine, a method, or a subprogram.

Example (Examples of specific tasks)

1. output data — e.g., `printf`
2. read data — e.g., `scanf`
3. Some more?

A simple example of function

Example (A simple example)

```
int abs(int a){
  if(a<0){
     a=a*(-1);
  }
  return a;
}
```

A simple example of function

Example (A simple example)

```c
int abs(int a){
  if(a<0){
     a=a*(-1);
  }
  return a;
}
```

Example (Call a function)

```c
int main(){
  int x;
  ...
  x=-2;
  printf("%d\n",abs(x));

}
```

What does `return` mean?

Remember that:

1. a function is a specific task

   e.g., to check if a number is even or odd.

2. after a function finishes, we hope it gives us some feedback

   e.g.
   - 1: odd
   - 0: even

What does `return` mean?

Remember that:

1. a function is a specific task

   e.g., to check if a number is even or odd.

2. after a function finishes, we hope it gives us some feedback

   e.g.
   - 1: odd
   - 0: even

- The feedback is returned by `return`.
- The value of a function call is the value of the expression following `return`.

Declare and define a function in C

Declaration of a function

```
Type1 functionName1();
Type1 functionName2(Type2 arg2, Type3 arg3,...);
Type1 functionName3(Type2, Type3,...);
```

- `Type*` is a type e.g., `int`, `char`.
- `arg*` is a variable, which is called argument.

Example

```
int abs(int a);
int abs(int);
```

If a function does not return anything, its type is `void`.

Declare and define a function in C

Declaration of a function

```
Type1 functionName1();
Type1 functionName2(Type2 arg2, Type3 arg3,...);
Type1 functionName3(Type2, Type3,...);
```

- `Type*` is a type e.g., `int`, `char`.
- `arg*` is a variable, which is called argument.

Example

```
int abs(int a);
int abs(int);
```

If a function does not return anything, its type is `void`.

Declaring a function is similar to declaring a variable!

Define a function

Defining of a function

```
Type functionName(Type1 arg1, Type2 arg2, ...){
  statements
}
```

Define a function

Defining of a function

```
Type functionName(Type1 arg1, Type2 arg2, ...){
  statements
}
```

Example (A simple example of defining a function)

```
int abs(int a){
  if(a<0){
    a=a*(-1);
  }
  return a;
}
```

Where to define a function

Define before the first function that calls it

Where to define a function

Define before the first function that calls it

Example

```c
#include <stdio.h>

float average(int i, int j){
  float f = (i+j)/2;
  return f;
}

int main(){
  int a=2,b=3;
  printf("Average of %d and %d is %f\n",a,b,average(a,b));
}
```

No need to declare

Where to define a function

Define after the first function that calls it

Where to define a function

Define after the first function that calls it

Example

```
#include <stdio.h>

float average(int,int);
^^I
int main(){
  int a=2,b=3;
  printf("Average of %d and %d is %f\n",a,b,average(a,b));
}

float average(int i, int j){
  float f = (i+j)/2;
  return f;
}
```

Must be declared before being called

Where to define a function

Define after the first function that calls it

Where to define a function

Define after the first function that calls it

Example

```
#include <stdio.h>
^^I
int main(){
  int a=2,b=3;
  float average(int,int); // this is also fine
  printf("Average of %d and %d is %f\n",a,b,average(a,b));
}
^^I
float average(int i, int j){
  float f = (i+j)/2;
  return f;
}
```

Must be declared before being called

But not necessarily declare before `main` function

Where to define a function

Define after the first function that calls it

Where to define a function

Define after the first function that calls it

Example

```c
#include <stdio.h>
int main(){
  int a=2,b=3;
  float average(int,int); // this is also fine
  printf("Average of %d and %d is %f\n",a,b,average(a,b));
}
float square(int i, int j){
  float s=average(i,j); // You need to declare average again here
  return s*s;
}^^I
float average(int i, int j){
  float f = (i+j)/2;
  return f;
}
```

Must be declared before being called

Where to define a function

Define in another file

Where to define a function

Define in another file

Example

```c
#include <stdio.h>
#include "average.h"
int main(){
  int a=2,b=3;
  printf("Average of %d and %d is %f\n",a,b,average(a,b));
}
```

Where to define a function

Define in another file

Example

```c
#include <stdio.h>
#include "average.h"
int main(){
  int a=2,b=3;
  printf("Average of %d and %d is %f\n",a,b,average(a,b));
}
```

Example (files: average.h and average.c)

```c
float average(int,int);
```

average.h

Where to define a function

Define in another file

Example

```c
#include <stdio.h>
#include "average.h"
int main(){
  int a=2,b=3;
  printf("Average of %d and %d is %f\n",a,b,average(a,b));
}
```

Example (files: average.h and average.c)

```c
float average(int,int);
```

average.h

```c
float average(int i, int j){
  float f = (i+j)/2;
  return f;
}
```

average.c

The mechanism of calling a function

Question: What happens when a function is called?

Example (When calling average at Line 8)

```
1 #include <stdio.h> ^^I
2 float average(int i, int j){
3   float f = (i+j)/2;
4   return f;
5 }
6 int main(){
7   int a=2,b=3;
8   printf("Average of %d and %d is %f\n",a,b,average(a,b));
9 }
```

- save a and b

The mechanism of calling a function

Question: What happens when a function is called?

Example (When calling average at Line 8)

```
1 #include <stdio.h> ^^I
2 float average(int i, int j){
3   float f = (i+j)/2;
4   return f;
5 }
6 int main(){
7   int a=2,b=3;
8   printf("Average of %d and %d is %f\n",a,b,average(a,b));
9 }
```

- save a and b
- declare i and j

The mechanism of calling a function

Question: What happens when a function is called?

Example (When calling average at Line 8)

```
1 #include <stdio.h> ^^I
2 float average(int i, int j){
3   float f = (i+j)/2;
4   return f;
5 }
6 int main(){
7   int a=2,b=3;
8   printf("Average of %d and %d is %f\n",a,b,average(a,b));
9 }
```

- save a and b
- declare i and j
- assign a to i and b to j

The mechanism of calling a function

Question: What happens when a function is called?

Example (When calling average at Line 8)

```
1 #include <stdio.h> ^^I
2 float average(int i, int j){
3   float f = (i+j)/2;
4   return f;
5 }
6 int main(){
7   int a=2,b=3;
8   printf("Average of %d and %d is %f\n",a,b,average(a,b));
9 }
```

- save a and b
- declare i and j
- assign a to i and b to j
- execute from Line 3

The mechanism of calling a function

Question: What happens when a function is called?

Example (When calling average at Line 8)

```
1 #include <stdio.h> ^^I
2 float average(int i, int j){
3   float f = (i+j)/2;
4   return f;
5 }
6 int main(){
7   int a=2,b=3;
8   printf("Average of %d and %d is %f\n",a,b,average(a,b));
9 }
```

- save a and b
- declare i and j
- assign a to i and b to j
- execute from Line 3
- delete i, j, f

The mechanism of calling a function

Question: What happens when a function is called?

Example (When calling average at Line 8)

```
1 #include <stdio.h> ^^I
2 float average(int i, int j){
3   float f = (i+j)/2;
4   return f;
5 }
6 int main(){
7   int a=2,b=3;
8   printf("Average of %d and %d is %f\n",a,b,average(a,b));
9 }
```

- save a and b
- declare i and j
- assign a to i and b to j
- execute from Line 3
- delete i, j, f

Why do we need to know the mechanism?

Example (Swap the values of two variables)

```
#include <stdio.h> ^^I
void swap(int i, int j){
  int tmp=j;
  j=i;
  i=tmp;
}
int main(){
  int a=2,b=3;
  swap(a,b);
  printf("%d,%d",a,b);
  return 0;
}
```

Why do we need to know the mechanism?

Example (Swap the values of two variables)

```
#include <stdio.h> ^^I
void swap(int i, int j){
  int tmp=j;
  j=i;
  i=tmp;
}
int main(){
  int a=2,b=3;
  swap(a,b);
  printf("%d,%d",a,b);
  return 0;
}
```

The output of executing the above code: ??

Why do we need to know the mechanism?

Example (Swap the values of two variables)

```
#include <stdio.h> ^^I
void swap(int i, int j){
  int tmp=j;
  j=i;
  i=tmp;
}
int main(){
  int a=2,b=3;
  swap(a,b);
  printf("%d,%d",a,b);
  return 0;
}
```

The output of executing the above code: 2,3

Why do we need to know the mechanism?

Example (Swap the values of two variables)

```
#include <stdio.h> ^^I
void swap(int i, int j){
  int tmp=j;
  j=i;
  i=tmp;
}
int main(){
  int a=2,b=3;
  swap(a,b);
  printf("%d,%d",a,b);
  return 0;
}
```

The output of executing the above code: 2,3

Why?

When calling swap at Line 9

Example (When calling swap at Line 8)

```
1 #include <stdio.h>
2 void swap(int i, int j){
3   int tmp=j;
4   j=i;
5   i=tmp;}
6 int main(){
7   int a=2,b=3;
8   swap(a,b);
9   printf("%d,%d",a,b);}
```

When calling swap at Line 9

Example (When calling swap at Line 8)

```
1 #include <stdio.h>
2 void swap(int i, int j){
3   int tmp=j;
4   j=i;
5   i=tmp;}
6 int main(){
7   int a=2,b=3;
8   swap(a,b);
9   printf("%d,%d",a,b);}
```

1. save a and b

When calling swap at Line 9

Example (When calling swap at Line 8)

```
1 #include <stdio.h>
2 void swap(int i, int j){
3   int tmp=j;
4   j=i;
5   i=tmp;}
6 int main(){
7   int a=2,b=3;
8   swap(a,b);
9   printf("%d,%d",a,b);}
```

1. save a and b
2. declare i and j

When calling swap at Line 9

Example (When calling swap at Line 8)

```
1 #include <stdio.h>
2 void swap(int i, int j){
3   int tmp=j;
4   j=i;
5   i=tmp;}
6 int main(){
7   int a=2,b=3;
8   swap(a,b);
9   printf("%d,%d",a,b);}
```

1. save a and b
2. declare i and j
3. assign a to i and b to j

When calling swap at Line 9

Example (When calling swap at Line 8)

```
1 #include <stdio.h>
2 void swap(int i, int j){
3   int tmp=j;
4   j=i;
5   i=tmp;}
6 int main(){
7   int a=2,b=3;
8   swap(a,b);
9   printf("%d,%d",a,b);}
```

1. save a and b
2. declare i and j
3. assign a to i and b to j
4. tmp=3 (Line 3), j=2(Line 4), i=3(Line 5)

When calling swap at Line 9

Example (When calling swap at Line 8)

```
1 #include <stdio.h>
2 void swap(int i, int j){
3   int tmp=j;
4   j=i;
5   i=tmp;}
6 int main(){
7   int a=2,b=3;
8   swap(a,b);
9   printf("%d,%d",a,b);}
```

1. save a and b
2. declare i and j
3. assign a to i and b to j
4. tmp=3 (Line 3), j=2(Line 4), i=3(Line 5)
5. Line 6: delete i,j, tmp

When calling swap at Line 9

Example (When calling swap at Line 8)

```
1 #include <stdio.h>
2 void swap(int i, int j){
3   int tmp=j;
4   j=i;
5   i=tmp;}
6 int main(){
7   int a=2,b=3;
8   swap(a,b);
9   printf("%d,%d",a,b);}
```

1. save a and b
2. declare i and j
3. assign a to i and b to j
4. tmp=3 (Line 3), j=2(Line 4), i=3(Line 5)
5. Line 6: delete i,j, tmp
6. restore a and b, and go to Line 9;

Summary

1. The notion of function
2. How to declare a function
3. How to define a function
4. Where to define a function
5. The mechanism of calling a function

Your assignment

☞ search and find a solution to the `swap` problem