Introduction to C Programming Language
PART II
Lecture 08

Min Zhang

zhangmin@sei.ecnu.edu.cn

2020.11.23

Software Engineering Institute

10:00-11:40, Monday, Room 319
Software Engineering Institute, East China Normal University

Review of Part I

What we have learned in Part I:

1. Types, operators, expressions
2. Statements, control flows
3. Functions
4. Array

Review of Part I

What we have learned in Part I:

1. Types, operators, expressions
2. Statements, control flows
3. Functions
4. Array

What we will learn in Part II:

1. Scope of variables
2. Macro
3. Pointers
4. Arguments of functions
5. Structure
6. I/O of files

Today's content

☞ **Scope of variables and Macros**

Recall the swap function

```c
#include <stdio.h>
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;
}
```

A (bad) solution

```c
#include <stdio.h>
int a=2,b=3;  // put a and b outside
void swap(){
  int tmp=a;
  a=b;
  b=tmp;
}
int main(){
  swap();
  printf("%d,%d",a,b);
  return 0;
}
```

A (bad) solution

```c
#include <stdio.h>
int a=2,b=3;  // put a and b outside
void swap(){
  int tmp=a;
  a=b;
  b=tmp;
}
int main(){
  swap();
  printf("%d,%d",a,b);
  return 0;
}
```

- When a function needs to *return multiple variables*, these variable should be put outside functions.
- These variables are called external variables.

External variables

Definition (External variables)

External variables are those declared outside any functions.

Why do we need external variables?

External variables can be read and written (referred to) by all the functions defined after them.

A simple example of using external variables

```c
int incTemp(int);
int decTemp(int);
int tmp = 0;
int main(){
  printf("%d\n",incTemp(3));
  printf("%d\n",decTemp(2));
  return 0;
}
int incTemp(int d){
  tmp = tmp+d ;
  return tmp;
}
int decTemp(int d){
  tmp = tmp-d ;
  return tmp;
}
```

A simple example of using external variables

```c
int incTemp(int);
int decTemp(int);
int tmp = 0;
int main(){
  printf("%d\n",incTemp(3));
  printf("%d\n",decTemp(2));
  return 0;
}
int incTemp(int d){
  tmp = tmp+d ;
  return tmp;
}
int decTemp(int d){
  tmp = tmp-d ;
  return tmp;
}
```

✎ Try by yourself!

A new conception: scope of functions or variables

Definition (Scope)

The scope of a variable or a function is the place where code can access it.

A new conception: scope of functions or variables

Definition (Scope)

The scope of a variable or a function is the place where code can access it.

```c
int incTemp(int);
int decTemp(int);
int main(){
  int tmp = 0;
  printf("%d\n",incTemp(3));
  printf("%d\n",decTemp(2));
  return 0;
}
int incTemp(int d){
  tmp = tmp+d ; // compile error here!!!
  return tmp;
}
int decTemp(int d){
  tmp = tmp-d ;
  return tmp;
}
```

Scope of an external variable

The scope of an external variable or a function lasts from the point where it is declared to the end of the file being complied.

Scope of an external variable

The scope of an external variable or a function lasts from the point where it is declared to the end of the file being complied.

```c
int incTemp(int);
int decTemp(int);
int main(){
  printf("%d\n",incTemp(3));
  printf("%d\n",decTemp(2));
  return 0;
}
int tmp = 0; // It is OK?
int incTemp(int d){
  tmp = tmp+d ;
  return tmp;
}
int decTemp(int d){
  tmp = tmp-d ;
  return tmp;
}
```

External variables declared in other files

```c
// main.c
#include <stdio.h>
#include "temp.h"
int tmp;
int main(){
  tmp=0; // initialize tmp
  printf("%d\n",incTemp(3));
  printf("%d\n",decTemp(2));
  return 0;
}^^I
```

```c
// temp.h
int incTemp(int);
int decTemp(int);

// temp.c
int incTemp(int d){
  tmp = tmp+d ;
  // compie error
  return tmp;
}
int decTemp(int d){
  tmp = tmp-d ;
  return tmp;
}
```

External variables declared in other files

```c
// main.c
#include <stdio.h>
#include "temp.h"
int tmp;
int main(){
  tmp=0; // initialize tmp
  printf("%d\n",incTemp(3));
  printf("%d\n",decTemp(2));
  return 0;
}
```

```c
// temp.h
int incTemp(int);
int decTemp(int);
extern int tmp;

// temp.c
int incTemp(int d){
  tmp = tmp+d;
  // compie error
  return tmp;
}
int decTemp(int d){
  tmp = tmp-d ;
  return tmp;
}
```

✎ Use keyword `extern` to declare those external variables that are declared in other files.

Static variable

Declaration of a static variable

```
static type variableName;
```

Static variable

Declaration of a static variable

```
static type variableName;
```

Scope of static variable

Static external variables are only visible to the rest of the source file being complied! Other files cannot access them.

- Static external variables: only in the file where it is declared!
- Static local variables: only initialized once!!

Static local variable

Local variable

Variables that are declared in functions.

Static local variable

Static variables that are declared in functions.

Static local variable

Local variable

Variables that are declared in functions.

Static local variable

Static variables that are declared in functions.

```c
int main(){
  int i;
  for(i=0;i<=100;i++){
    int sum=0;
    sum=sum+i;
    if(i==100){
      printf("%d",sum);
    }
  }
  return 0;
}
```

```c
int main(){
  int i;
  for(i=0;i<=100;i++){
    static int sum=0;
    sum=sum+i;
    if(i==100){
      printf("%d",sum);
    }
  }
  return 0;
}
```

More example on static local variables

Example

```c
#include <stdio.h>

int inc(){
 static int counter=0; // the times of inc() call
 counter++;
 return counter;
}
^^I
int main(){
  printf("%d\n",inc());
  printf("%d\n",inc());
  return 0;
}
```

✎ Try without `static`!

Variables with the same name

It is allowed that variables have the same name and the same type.

Variables with the same name

It is allowed that variables have the same name and the same type.

Example

```c
#include <stdio.h>
int main(){
  int i=1;
  int j=2;
  {
    int i=2+j;
    printf("i=%d\n",i);
  }
  printf("i=%d\n",i);
  return 0;
}
```

Variables with the same name

It is allowed that variables have the same name and the same type.

Example

```c
#include <stdio.h>
int main(){
  int i=1;
  int j=2;
  {
    int i=2+j;
    printf("i=%d\n",i);
  }
  printf("i=%d\n",i);
  return 0;
}
```

```
Output:
i=4
i=1
```

Variables with the same name

✎ Internal variables live only in the block where they are declared!

Initialization of variables

✎ Intialization occurs when variales are declared and meanwhile set values!

Rules for initializing variables:

- For external and static values, expressions must not contain variables.
- For internal variables, expressions can have variables and even function calls. Initialization is done each time the function or block is entered.

```
static int x = y+1; // BAD!!
```

Register variable

A special variable which is stored in registers in computer.

```
register Type variableName;
```

✎ Used when variables are frequently accessed!

C preprocessing

Process before compiling

1. `#include`
2. `#define`
3. `#if #elif #else #endif`

#include

- `#include <filename>`
  to search file following an implementation-defined rule
- `#include "filename"`
  to search from where the source program was found

#define

```
#define name replacement text
```

Example

```
#define max(A,B) ((A) > (B) ? (A) : (B))
```

`x = max(p+q,r+s);` would be

`x = ((p+q) > (r+s) ? (p+q) : (r+s));`

before being compiled.

#define

```
#define name replacement text
```

Example

```
#define max(A,B) ((A) > (B) ? (A) : (B))
```

`x = max(p+q,r+s);` would be

`x = ((p+q) > (r+s) ? (p+q) : (r+s));`

before being compiled.

A test

```
int i=3,j=4;
int m=max(i++,j++);
printf("%d\t%d\t%d\n",i,j,m);
```

Output:

#define

```
#define name replacement text
```

Example

```
#define max(A,B) ((A) > (B) ? (A) : (B))
```

`x = max(p+q,r+s);` would be

`x = ((p+q) > (r+s) ? (p+q) : (r+s));`

before being compiled.

A test

```
int i=3,j=4;
int m=max(i++,j++);
printf("%d\t%d\t%d\n",i,j,m);
```

Output: 4 6 5

#define

```
#define name replacement text
```

Example

```
#define max(A,B) ((A) > (B) ? (A) : (B))
```

`x = max(p+q,r+s);` would be

`x = ((p+q) > (r+s) ? (p+q) : (r+s));`

before being compiled.

A test

```
int i=3,j=4;
int m=max(i++,j++);
printf("%d\t%d\t%d\n",i,j,m);
```

Output: `4 6 5`

That is because `m = ((i++) > (j++) ? (i++) : (j++))`

#if #elif #else #endif

Conditional inclusion

To control preprocessing itself while conditional statements that are evaluated during preprocessing!

To include different file based on the type of OS

```
#if SYSTEM == SYSV
  #define HDR "sysv.h"
#elif SYSTEM == BSD
  #define HDR "bsd.h"
#elif SYSTEM == MSDOS
  #define HDR "msdos.h"
#else
  #define HDR "default.h"
#endif
#include HDR
```

Summary

1. External variables
2. Static variables
3. Scope of variables
4. Register variables
5. Macro