Structured Programming

- Variable scope and parameters

Donglong Chen

With Thanks to Dr. Xin Feng and Dr. Haipeng Guo

Outline

- Global variables
- Local variables
- Perspective of a program
- Actual parameters and formal parameters
- Parameter passing

Scope

- The scope of a declaration is the block of code where the identifier is valid for use.
- A global declaration is made outside the bodies of all functions and outside the main program.
  - It is normally placed at the beginning of the program file.
- A local declaration is one that is made inside the body of a function.
  - Locally declared variables cannot be accessed outside of the functions where they were declared.
- It is possible to declare the same identifier name in different parts of a program.

Scope

```
int y = 38; // global variable
int f(int, int);
void main( )
{
  int z = 47;
  while(z < 400){
    int a = 90;
    z += a++;
    z++;
  }
  y = 2 * z;
  f(1, 2);
}
int f(int s, int t)
{
  int r;
  r = s + t;
  int i = 27;
  r += i;
  return r;
}
```

scope of y

scope of f

scope of z

scope of a

scope of s & t

scope of r

scope of i

Class Exercise #1

```
int number; //number: global variable
void increment(int num) // num: local variable
{
  num = num + 1;
  printf("%d\n", num);
  number = number + 1;
}
void main()
{
  number = 1;
  increment(number);
  printf("%d\n", number);
}
```

What is the value of number?

Class Exercise #2

```
int number; //number: global variable
void increment(int number) // number: local variable
{
  number = number + 1; //use the local number
}
void main()
{
  number = 1;  //use the global number
  increment(number); //use the global number
  printf("%d\n", number); //use the global number
}
```

What is the value of number?

Disadvantage of Global Variables

- Undisciplined use of global variables may lead to confusion and debugging difficulties.
- To pass the values to formal parameters, we can pass address (or reference) to formal parameters so that later more values can be passed back to main function.

Value Passing

- A function call can return only one single result

```
#include <stdio.h>
int sum(int, int);
int main()
{
   int o1, o2;
   scanf("%d %d", &o1, &o2);
   printf("The sum is %d", sum(o1, o2));
   printf("2 + 3 = %d", sum(2, 3));
   return 0;
}
int sum(int operand1, int operand2)
{
   return (operand1 + operand2);
}
```

How about if we want multiple values to be returned to the main function?

Value Passing

```
double calSumAverage(double, double );
int main( )
{
  double x = 1.0, y = 2.0;
  printf("The sum is %f",calSumAverage(x, y));
  return 0;
}
double calSumAverage(double no1, double no2)
{
    double sum, average;
    sum = no1 + no2;
    average = sum / 2;
    return sum;
}
```

We can return either sum or average, but not both

Actual and Formal Parameters

```
#include <stdio.h>
int sum(int, int);
int main()
{
   int o1, o2;
   scanf("%d %d", &o1, &o2);
   printf("The sum is %d", sum(o1, o2));
   return 0;
}
int sum(int operand1, int operand2)
{
   return (operand1 + operand2);
}
```

Actual parameters

Formal parameters

Parameter Passing

- We can pass values through parameters
- In actual parameters, we can not only pass the values, but also addresses.

Parameter Passing-Class Exercise #3

- Compare the following three examples, what is the value of *k* after the function call?

```
int main(void)
{
  int k = 10;
  foo(k);
  printf("%d", k);
}
void foo(int j)
{
  j = 0;
}
```

```
int main(void)
{
  int k = 10;
  foo(k);
  printf("%d", k);
}
void foo(int k)
{
  k = 0;
}
```

```
int main(void)
{
  int k = 10;
  foo(&k);
  printf("%d", k);

}
void foo(int *j)
{
  *j = 0;
}
```

Parameter Passing

- In first two example2, the parameter's value is passed.
  - all information in local variables declared within the function will be lost when the function terminates
- In the third example, the parameter's address is passed
  - In this way, any changes to formal parameter will affect the values of actual parameter variables
  - More details about address will be given in the "Pointer" lecture.

Parameter Passing

- Compare the following three examples, what is the value of *k* after the function call?

```
int main(void)
{
  int k = 10;
  foo(k);
}
void foo(int j)
{
  j = 0;
}
```

Pass by value

```
int main(void)
{
  int k = 10;
  foo(k);
}
void foo(int k)
{
  k = 0;
}
```

Pass by value

```
int main(void)
{
  int k = 10;
  foo(&k);
}
void foo(int *j)
{
  *j = 0;
}
```

Pass by address

Parameter Passing

```c
#include<stdio.h>
void foo(int *j);
int main(void)
{
  int k = 10;
  printf("%d\n",k);
  printf("%x\n",&k);
  foo(&k);
}
void foo(int *j)
{
  *j = 0;
  printf("%d\n",*j);
  printf("%x\n",j);
}
```

C:\windows\system32\cmd.exe

```
10
affe54
0
affe54
请按任意键继续. . .
```

Address	Data
0xaffe51	
0xaffe52	
0xaffe53	
0xaffe54	10
0xaffe55	
0xaffe56	

Value Passing

```
double calSumAverage(double, double );
int main( )
{
  double x = 1.0, y = 2.0;
  printf("The sum is %f", calSumAverage(x, y));
  return 0;
}
double calSumAverage(double no1, double no2)
{
    double sum, average;
    sum = no1 + no2;
    average = sum / 2;
    return sum;
}
```

We can return either sum or average, but not both.

However, we can use address to pass both back to the main function. Will be introduced later

Pass Arrays to Functions Passing-Class Exercise #4

```
void exchange(float a[], int n); // function prototype

int main(){
    float value[4] = {2.5, -4.75, 1.2, 3.67};
    exchange(value, 4);
    printf("value[0] = %f", value[0]);
    return 0;
}

void exchange(float a[], int n){
    float temp;
    temp = a[0];
    a[0] = a[n - 1];
    a[n - 1] = temp;
}
```

What is the output value?

Pass Arrays to Functions

- Any changes in the array in the called function will be reflected in the original array

Placement of Functions

- For a small program, use the following order in the only one file:

.c

1. `include directives`
2. `all function prototypes`
3. `main() function`
4. `Other function definitions`

Placement of Functions

- For large programs
  - Manage related functions in .c files
  - Write .h files containing all the prototypes of the functions
  - Include the header files in the files that use the functions.
    - `#include "mymath.h"`

mymath.h

```
int min(int x,int y);
int max(int x,int y);
```

mymath.c

```
int min(int x,int y)
{
  return x > y? y: x
}

int max(int x,int y)
{
  return x > y? x: y;
}
```

Perspective of a Program

- A program is comprised of functions (logically).
- A program is comprised of files (physically).

Summary

- A variable has its scope
- In a function, parameter passing can only pass values. The variable in the function call will not be affected.
- Functions in a program can be put in different files. But there is only one main function.