Functions (1)
Storage Class and Scope

Young W. Lim

December 9, 2017
1 Scope of variables

1.1 local variables

```
#include <stdio.h>

int func1(int a, int b)
{
    int result;
    printf("func1: &a= %p, a= %d\n", &a, a);
    printf("func1: &b= %p, b= %d\n", &b, b);
    result = (a+b);
    return (result);
}

int func2(int *m, int *n)
{
    int result;
    printf("func2: &m= %p, m= %p\n", &m, m);
    printf("func2: &n= %p, n= %p\n", &n, n);
    printf("func2: m= %p, *m= %d\n", m, *m);
    printf("func2: n= %p, *n= %d\n", n, *n);
    result = (*m+*n);
    return (result);
}

int main(void) {
    int a = 100;
    int b = 200;
    int S;
    printf("main: &a= %p, a= %d\n", &a, a);
    printf("main: &b= %p, b= %d\n", &b, b);
    S = func1 (a, b);
    printf("main: a=%d b=%d S=%d\n", a, b, S);
    S = func2 (&a, &b);
    printf("main: a=%d b=%d S=%d\n", a, b, S);
}
```
The addresses are not fixed. They change whenever the program is executed.

- these a’s are different because their memory locations are different.
  - the address of a in main() : 0x7ffc818bc16c
  - the address of a in func1() : 0x7ffc818bc13c
- the argument a is passed by reference to the parameter m
  - the address of a is copied into m
  - the value of a is accessed as *m
  - it can be read and written through *m
  - if a new value is written to *m
  - then a in main() also will have the same new value
- *m = 0
  - m has the address of a : 0x7ffe818bc16c
  - *m has the initial value of 100
  - *m will have the new value of zero (*m = 0)
  - the content at 0x7ffe818bc16c will be changed to zero
  - the value of a will be changed to zero
1.2 Static Local Variables

```c
#include <stdio.h>

int func1( void ) {
    int val = 0;
    val += 1;
    return (val);
}

int func2( void ) {
    static int val = 0;
    val += 1;
    return (val);
}

int main( void ) {
    int S = 0;
    S = func1(); printf("1st call func1 : S= %d \n", S);
    S = func1(); printf("2nd call func1 : S= %d \n", S);
    S = func1(); printf("3rd call func1 : S= %d \n", S);
    printf("----------------\n");
    S = func2(); printf("1st call func2 : S= %d \n", S);
    S = func2(); printf("2nd call func2 : S= %d \n", S);
    S = func2(); printf("3rd call func2 : S= %d \n", S);
}
```

```
1st call func1 : S= 1
2nd call func1 : S= 1
3rd call func1 : S= 1
----------------
1st call func2 : S= 1
2nd call func2 : S= 2
3rd call func2 : S= 3
```
val in func1

- a local variable
- the automatic storage class
- whenever `func1()` is called, `val` is created and initialized with 0.
- whenever `func1()` is returned, `val` is removed from the memory.

val in func2()

- a `static` local variable
- the static storage class
- when `func2()` is called for the first time, `val` is created and initialized with 0 once.
- even when `func2()` is returned, `val` is not removed but retains its current value.

### 1.3 Global Variables

```
t2.c

#include <stdio.h>

int i=999;

void func1( void ) {
    printf("func1: global i= %d \n", i);
}

void func2( void ) {
    int i = 0;

    printf("func2: local i= %d \n", i);
    i = 777;
}

void func3( void ) {
    i = 222;

    printf("func3: global i= %d \n", i);
}

int main( void ) {
    printf("main: global i= %d \n", i);
    printf("-----\n");
```
func1();
func2();
func3();

printf("main: global i= %d \n", i);
printf("--------------\n");

i = 333;
func1();
func2();
func3();

printf("main: global i= %d \n", i);
printf("--------------\n");
}

--------------------
t2.out
--------------------
main: global i= 999
--------------
func1: global i= 999
dfunc2: local i= 0
func3: global i= 222
main: global i= 222
--------------
func1: global i= 333
dfunc2: local i= 0
func3: global i= 222
main: global i= 222
--------------

func1

• the global variable i=999 is seen

func2

• its own local variable i hides the global variable i=999
• the local variable i=0 is seen
• though its value is changed to 777, it will be destroyed when func2() is returned

func3
• no local variable i defined
• the global variable i is accessed and set to 222
• the global variable i=222 is seen

main
• before i=333 statement
  – func1: global i= 999
  – func2: local i= 0
  – func3: global i= 222
  – main: global i= 222
• after i=333 statement
  – func1: global i= 333
  – func2: local i= 0
  – func3: global i= 222
  – main: global i= 222

1.4 Variable Scope

```c
#include <stdio.h>

int a = 1;
int b = 2;

void func(int a, int b) {
    a = 555;
    printf("a= %d  b=%d\n", a, b); // (1)
    b = 555;
}

void swap(int *a, int *b) {
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}

int main(void) {
    int a= 111;
```
int b = 222;
{
  int a = 888;
  printf("a = %d b = %d\n", a, b); // (2)
}

func(a, b);
printf("a = %d b = %d\n", a, b); // (3)
swap(&a, &b);
printf("a = %d b = %d\n", a, b); // (4)

in func()
  • in func(), parameter variable a and b are defined
  • they are also local variables of func
  • in main(), func is called with the argument values of 111 and 222
  • the local variable a is changed into 555
  • print the local variable a and b : 555, 222 : (1)
  • the local variable b is changed into 555
  • after returning, these local variables will be vanished

in main()
  • local variable a and b are declared and hide the global variable a and b
  • the nested block has also another local variable a with the value of 888
  • this new a hides the local variable a of main()
  • print a and b in this block displays the block’s local variable a (888) and main’s local variable b (222) : (2)
• after calling func(a,b), neither global nor main’s local a and b will be changed
• but the global variables a and b are covered by its local variables a (111) and b (222) \( : (3) \)
• after calling swap(a,b), the values of the local variable a and b are swapped : a (222) and b (111) : \( (4) \)