

ECE 220 Computer Systems & Programming

Lecture 9 – Functions in C & Run-Time Stack

September 24, 2019



C Functions

Provides abstraction

- hide low-level details
- give high-level structure to program, easier to understand overall program flow
- enable separable, independent development
- reuse code

Structure of a function

- zero or multiple arguments passed in
- single result returned (optional)
- return value is always a particular type

Making a Function Call in C

```
#include <stdio.h>
/* our Factorial function prototype goes here */
int Fact(int n);

/* main function */
int main() {
    int number;
    int answer;

    printf("Enter a number: ");
    scanf("%d", &number);

    answer = Fact(number); /* function call */
    /* number - argument transferred from main to Factorial */
    /* answer - return value from Factorial to main */

    printf("factorial of %d is %d\n", number, answer);

    return 0;
}
```

Function “Fact”:

```
/* implementation of Factorial function goes here */  
int Fact(int n) {  
    int i, result=1; /* local variables in Factorial */  
  
    for (i = 1; i <= n; i++)  
        result = result * i;  
  
    return result; /* return value */  
}
```

Function that does not return value:

```
1  #include <stdio.h>
2
3  void PrintBanner();      /* Function declaration */
4
5  int main()
6  {
7      PrintBanner();        /* Function call      */
8      printf("A simple C program.\n");
9      PrintBanner();
10 }
11
12 void PrintBanner()        /* Function definition */
13 {
14     printf("=====\n");
15 }
```

***Note:** Functions do not necessarily have to be in the same file
(see the github example)

print.h ---> declares the function prototype

main.c ---> call the “print” function

print.c ---> print function

Example:

```
1  #include <stdio.h>
2  void print_val(int x, int y);
3
4  void print_val(int x, int y)
5  {
6      printf("first value:%d second value:%d\n", x, y);
7  }
8
9  int main()
10 {
11     int z = 0;
12     printf("value of z++:%d\n", z++);
13     z = 0;
14     printf("value of ++z:%d\n", ++z);
15
16     int val = 10;
17     print_val(val, val++);
18
19     return 0;
20 }
```

How about the following “swap” function?

```
1  #include <stdio.h>
2  void swap(int x, int y);
3  int main()
4  {
5      int x = 1;
6      int y = 2;
7
8      printf("Before swap: x = %d, y = %d\n", x, y);
9      swap(x, y);
10
11     //Did the swap function work the way you expect?
12     printf("After swap: x = %d, y = %d\n", x, y);
13
14     return 0;
15 }
16
17 void swap(int x, int y)
18 {
19     int temp;
20
21     temp = x;
22     x = y;
23     y = temp;
24 }
```

Possible Solution

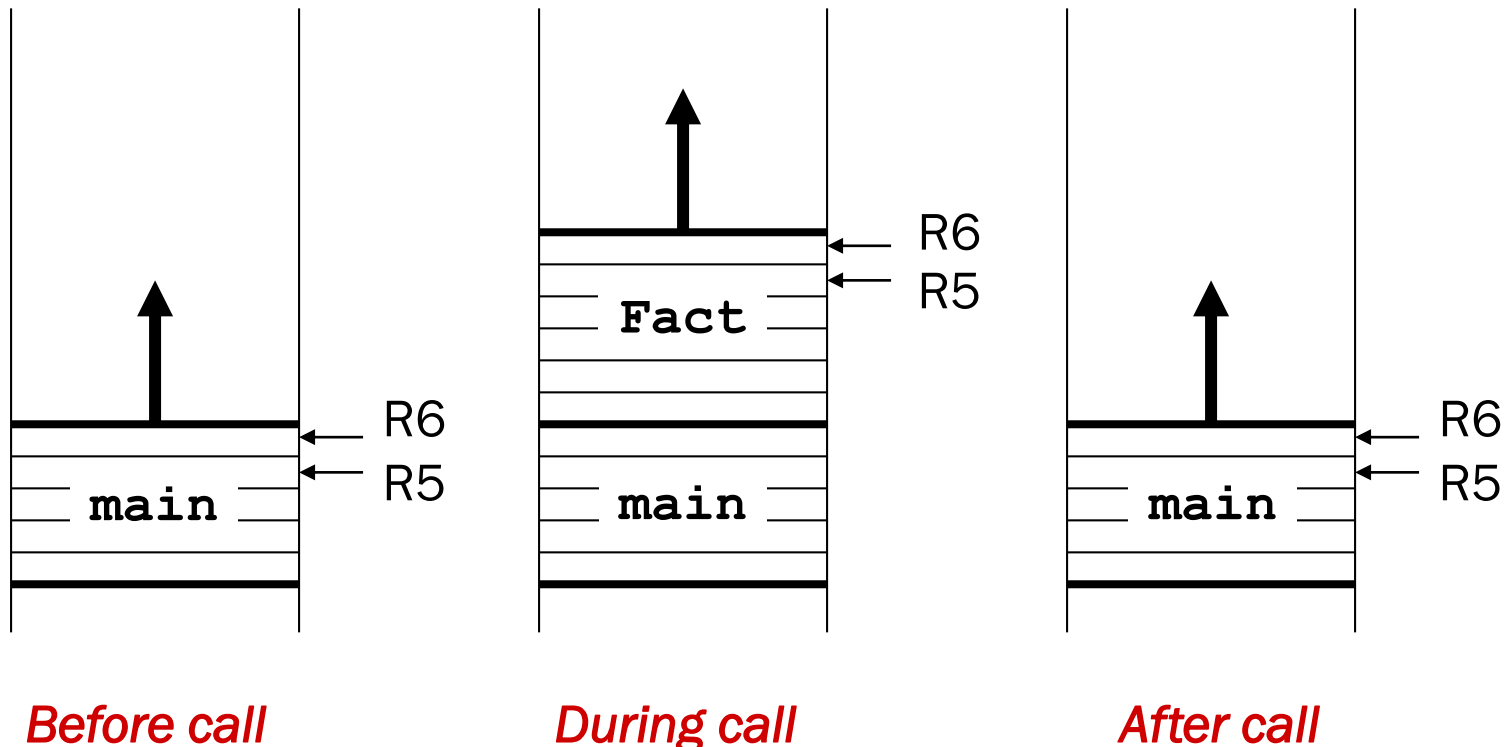
```
1  #include <stdio.h>
2  void swap(int x, int y);
3  int z, k;
4
5  int main()
6  {
7      int x = 1;
8      int y = 2;
9
10     printf("Before swap: x = %d, y = %d\n", x, y);
11     swap(x, y);
12
13     //Did the swap function work the way you expect?
14     printf("After swap: x = %d, y = %d\n", z, k);
15
16     return 0;
17 }
18
19 void swap(int x, int y)
20 {
21     int temp;
22
23     temp = x;
24     x = y;
25     y = temp;
26     z=x;
27     k=y;
28 }
```


Possible Solution (advanced topics coming soon!)

```
1  #include <stdio.h>
2
3  void swap(int *, int *);
4
5  int main()
6  {
7      int x = 1;
8      int y = 2;
9
10     printf("Before swap: x = %d, y = %d\n", x, y);
11     swap(&x, &y);
12
13     //Did the swap function work the way you expect?
14     printf("After swap: x = %d, y = %d\n", x, y);
15
16     return 0;
17 }
18
19 void swap(int *x, int *y)
20 {
21     int temp;
22
23     temp = *x;
24     *x = *y;
25     *y = temp;
26 }
```

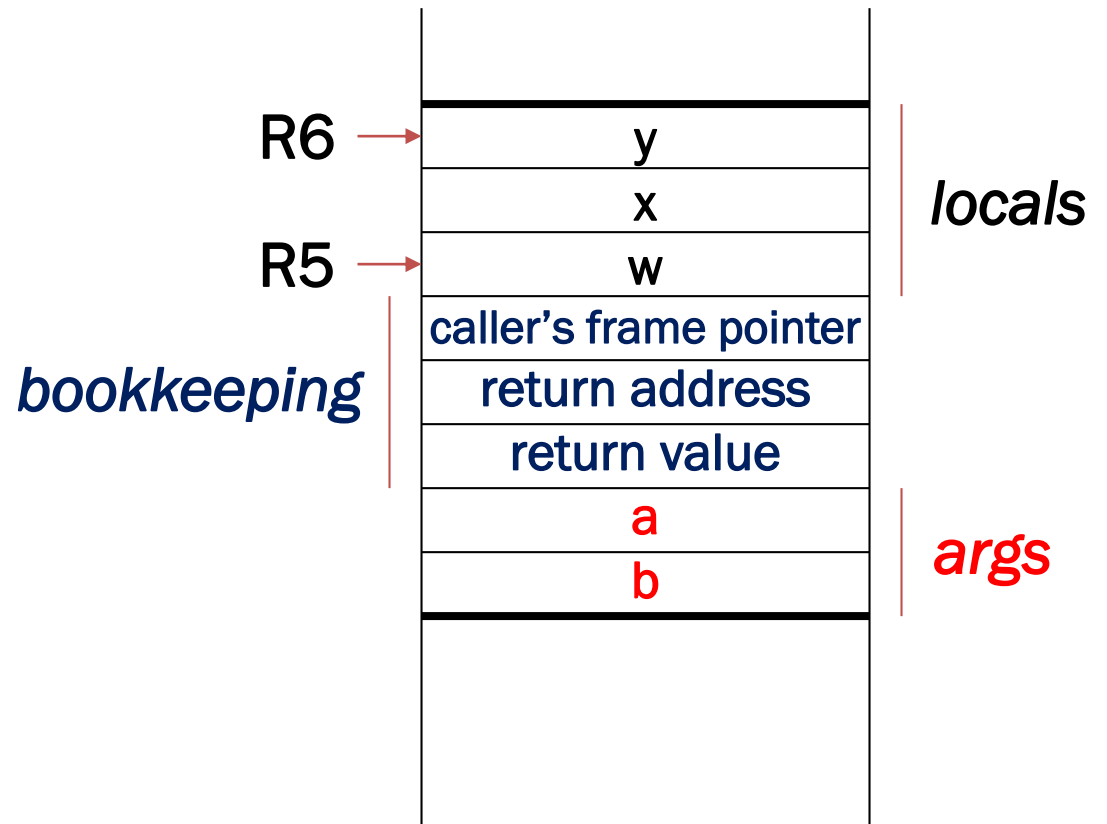
Run-Time Stack

- R5 – **Frame Pointer**. It points to the beginning of a region of activation record that stores local variables for the current function.
- R6 – **Stack Pointer**. It points to the top most occupied location on the stack.
- Arguments are pushed to the stack _____.
- Local variables are pushed to the stack _____.

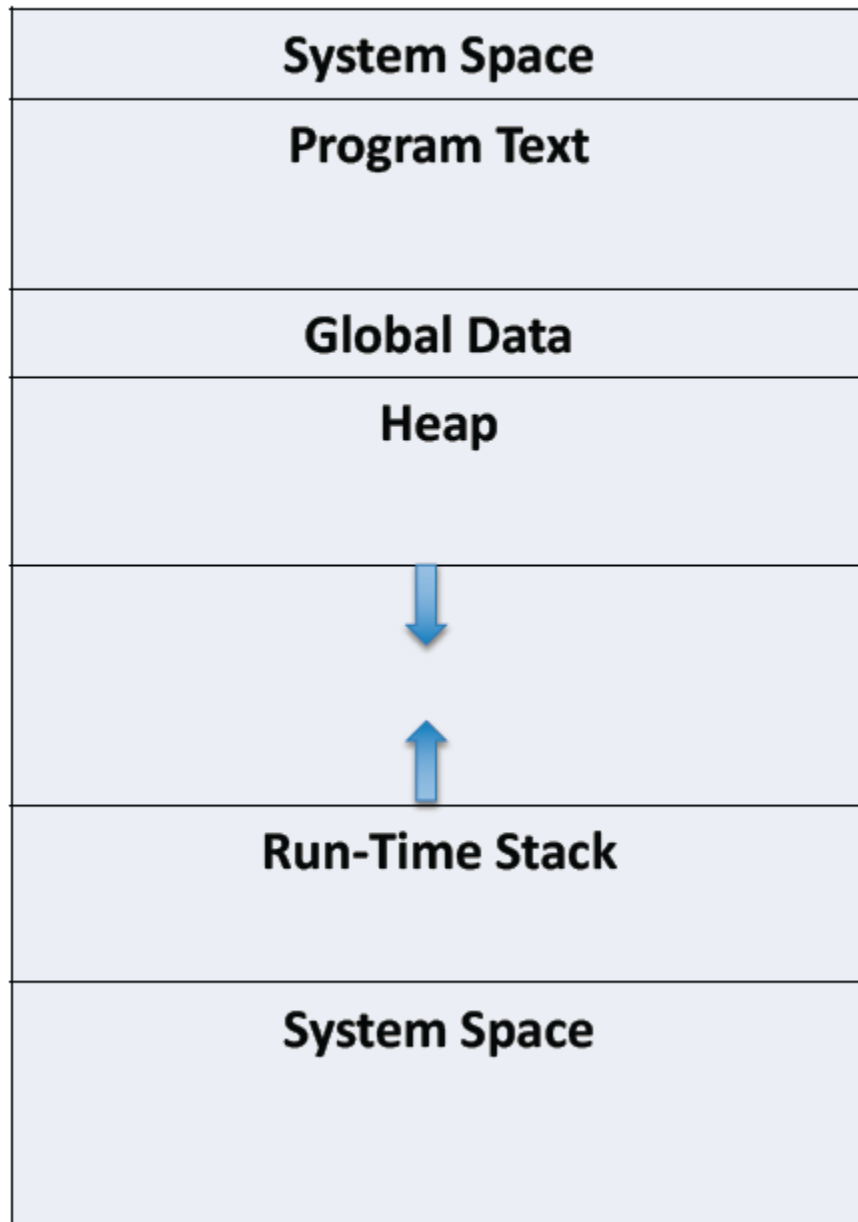


Activation Record

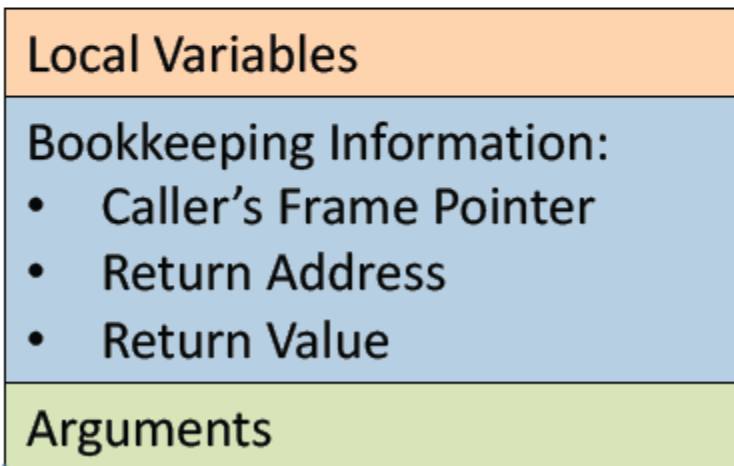
```
int func(int a, int b)
{
    int w, x, y;
    .
    .
    .
    return y;
}
```



LC-3 Memory Map



Activation Record



Stack Built-up and Tear-down

- | | | |
|-----------------|---|--|
| Caller function | { | 1. <u>caller setup</u> : push callee's arguments onto stack |
| | | 2. pass control to callee (invoke function) |
| Callee function | { | 3. <u>callee setup</u> : (push bookkeeping info and local variables onto stack) |
| | | 4. execute function |
| | | 5. <u>callee teardown</u> : (pop local variables, caller's frame pointer, and return address from stack) |
| | | 6. return to caller |
| Caller function | | 7. <u>caller teardown</u> : (pop callee's return value and arguments from stack) |