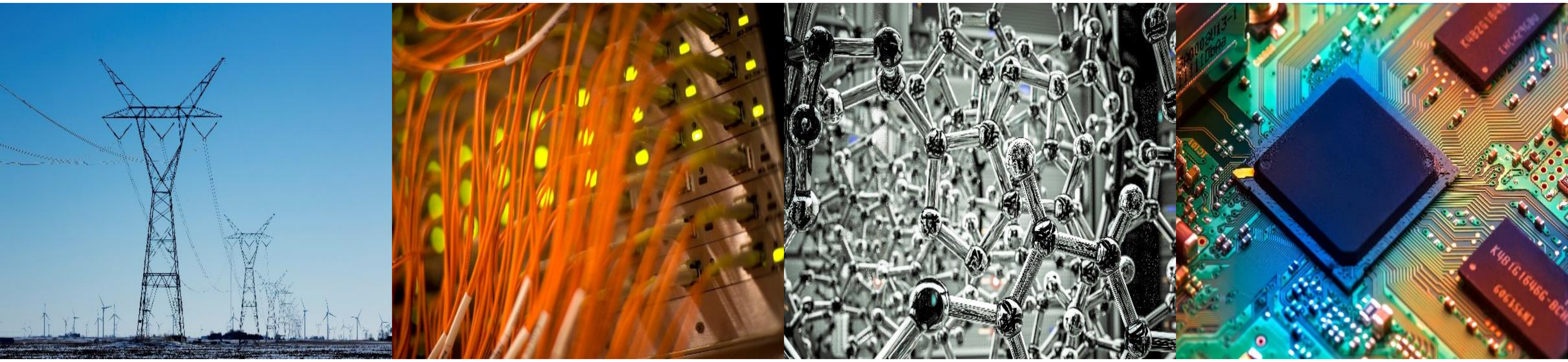
ECE 220 Computer Systems & Programming

Lecture 7 – Functions in C & Run-Time Stack

September 17, 2024



- Quiz1 should be completed at CBTF by Wednesday
- MT1 is next Thursday, 9/26; conflict sign-up deadline is 9/22
- HKN Review Session: Sunday, 9/22, 12:30 – 3PM, ECEB 1002

ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

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;
}
```

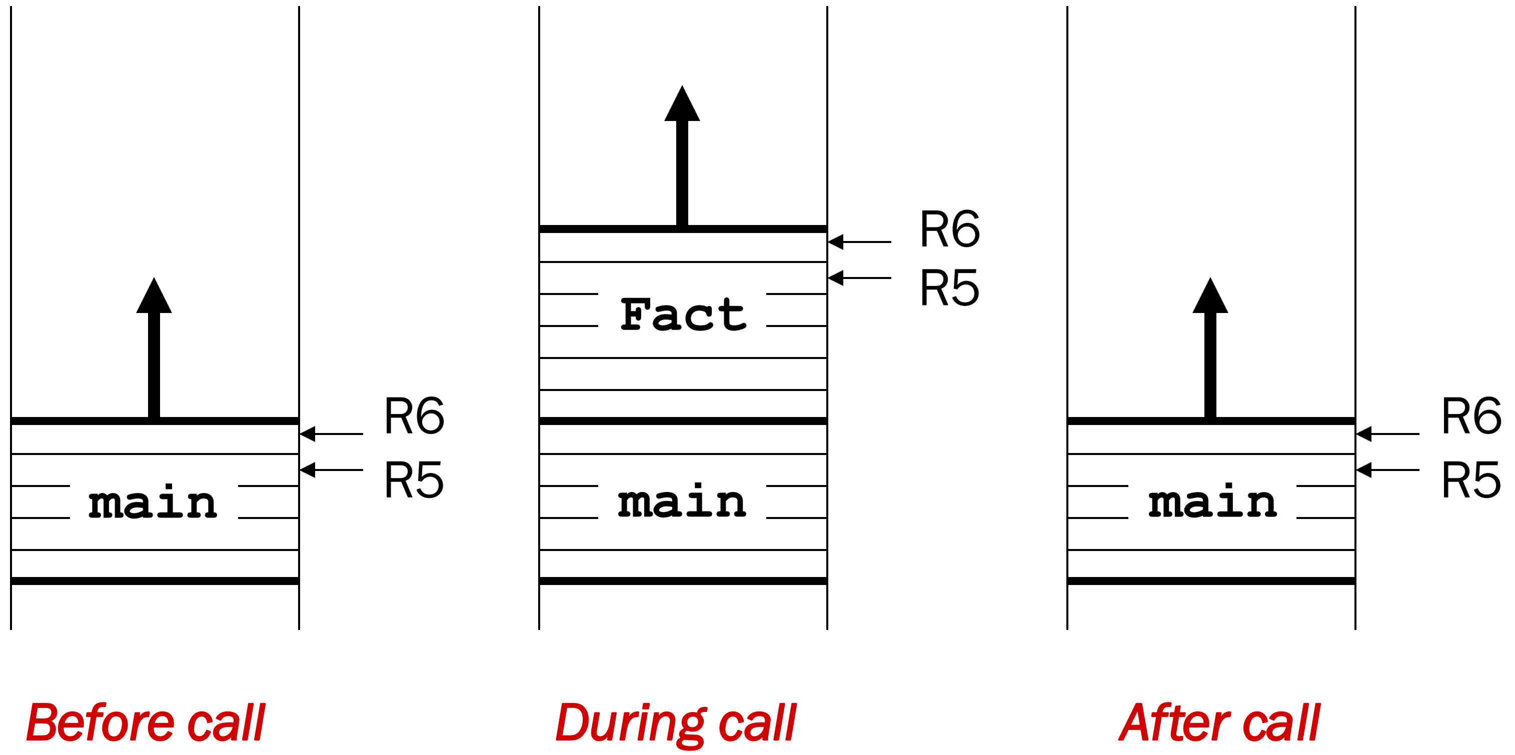
```
/* implementation of Factorial function goes here */
int Fact(int n) {
   int i, result=1; /* local variables in Factorial */

   for (i = 1; i <= n; i++)
      result = result * i;

   return result; /* return value */
}
```

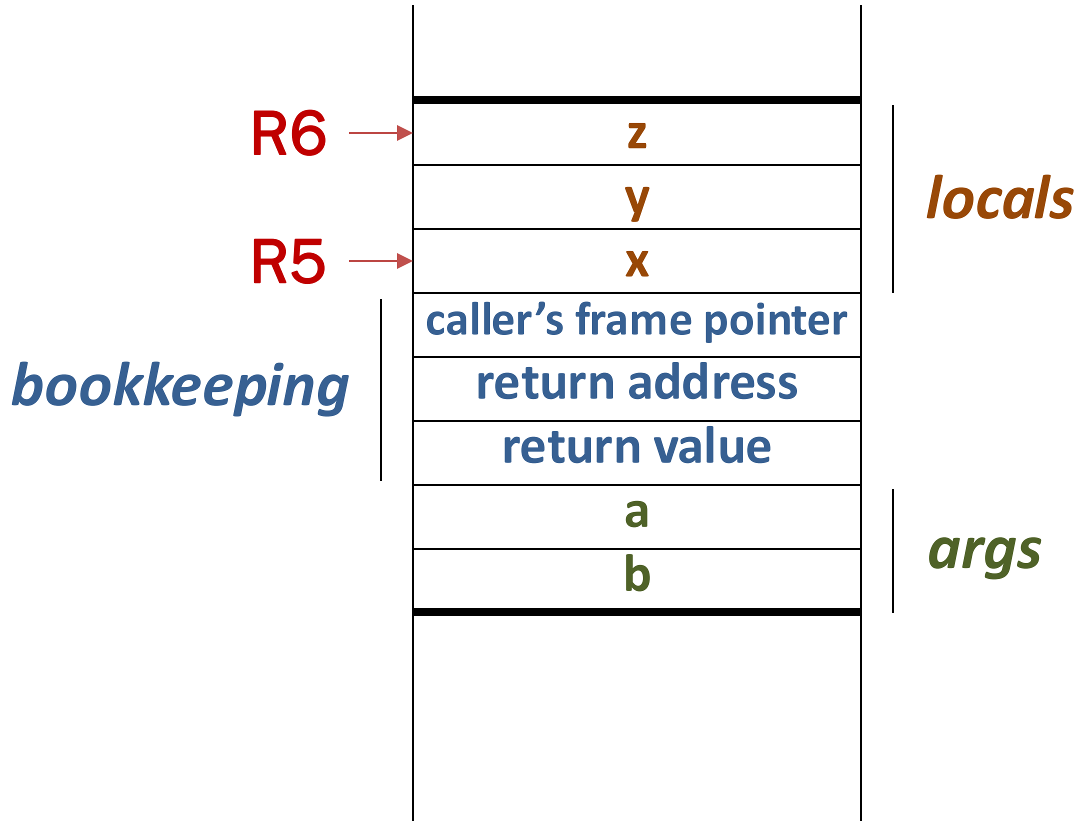
Run-Time Stack

- 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.
- In LC-3, arguments are pushed to the stack ___________________________, local variables are pushed to the stack ___________________________.

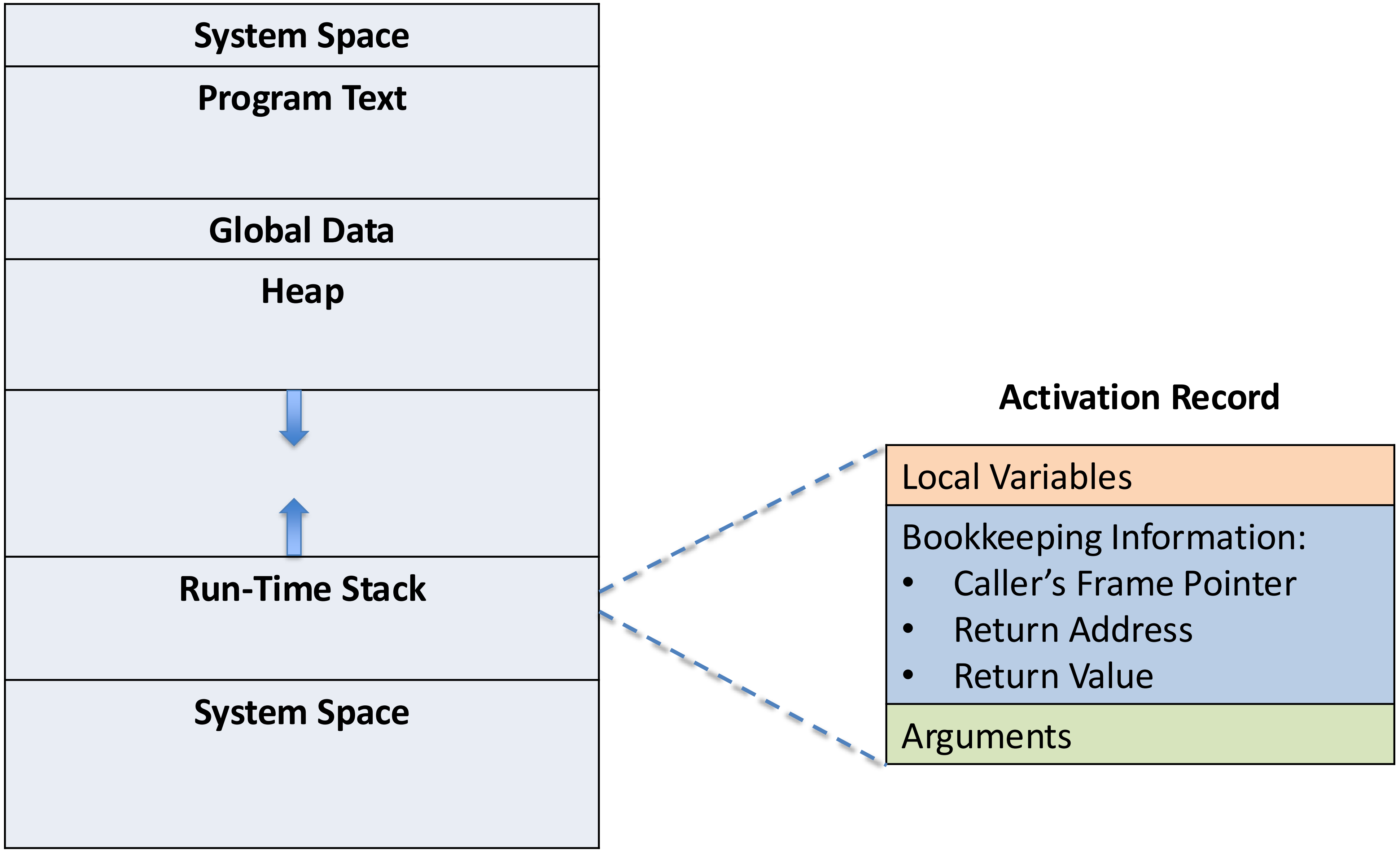


Activation Record (Stack Frame)

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



Memory Organization



Stack Built-up and Tear-down

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

Run-Time Stack Exercise

```
#include <stdio.h>
int Fact(int n);

int main() {
   int number;
   int answer;
   …
   answer = Fact(number);
   …
   return 0;
}

int Fact(int n) {
   int i, result=1;

   for (i = 1; i <= n; i++)
      result = result * i;

   return result;
}
```

	x3FF7	
	x3FF8	
	x3FF9	
	x3FFA	
	x3FFB	
	x3FFC	
	x3FFD	
	x3FFE	
main's activation record	x3FFF	answer
	x4000	number