# ECE 220 Computer Systems & Programming

## Lecture 9 – Functions in C & Run Time Stack

## September 26, 2017

- Midterm 1 scheduled on Thursday, 9/28
- Time & room assignment posted on Compass
- On the day of Midterm 1: lecture will be cancelled, no office hours between 5pm and 6pm
- MP3 is due Saturday, 9/30, at 10pm

ECE ILLINOIS

ILLINOIS

# Nested For Loop (from Lecture 8)

```
#include <stdio.h>
/* use nested for loops to print an n x n identity matrix */
int main(){
   int i, j, n=0; //i is the row index, j is the column index
   printf("Enter a number for nxn matrix size: ");
   scanf("%d", &n);
   printf("Output Identity Matrix: \n");
   for (i = 0; i < n; i++) { //outer loop – go through rows
        for (j = 0; j < n; j++) { //inner loop – go through columns
                if(i == j) //on major diagonal
                   printf("1");
                else
                   printf("0");
        }
        printf("\n"); //print new line after each row
   }
   return 0;
}
```

- What are some ways to stop after printing the second ’1’ on the main diagonal? (see example below)

  1 0 0

  0 1

- How to add a check before printing the matrix to ensure user input is within the valid range of 0<n<10? (If user input is invalid, print the message “Number entered is invalid” and prompt the user to enter a number again. )

# 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;
}
```

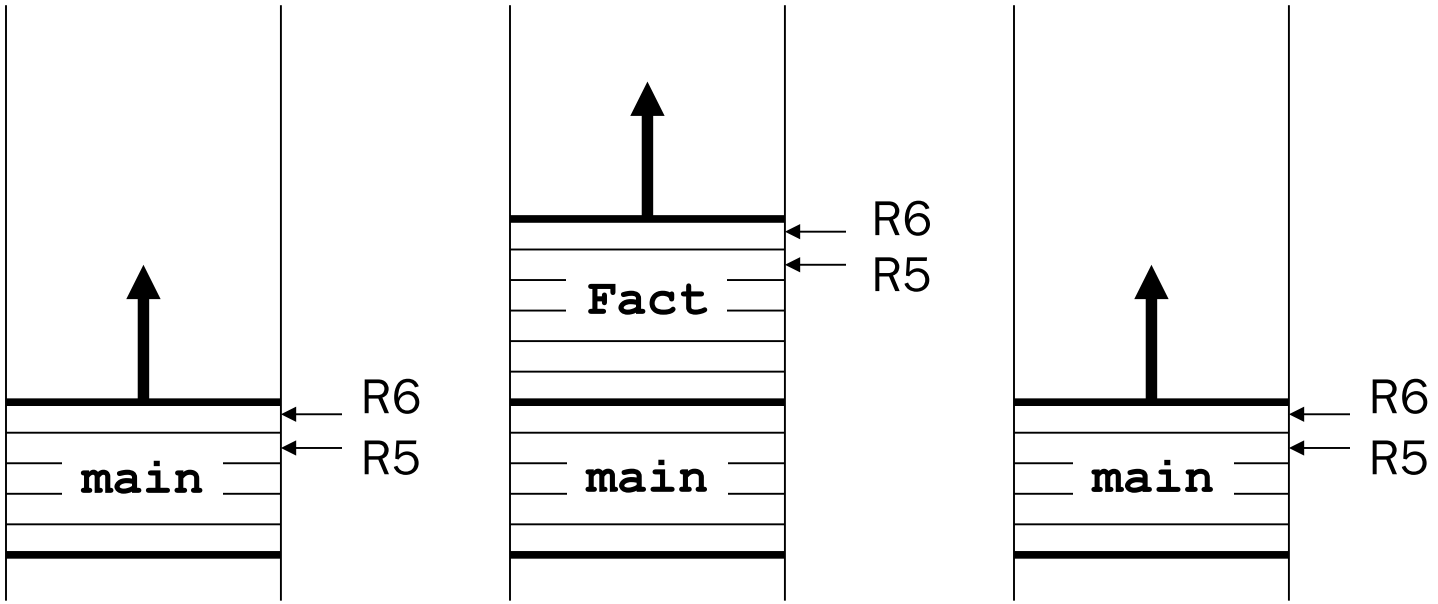
```
/* implementation of Factorial function goes here */
int Fact(int n) {
    int i, result=1; /* local variables in Factorial */

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

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

# Run Time Stack

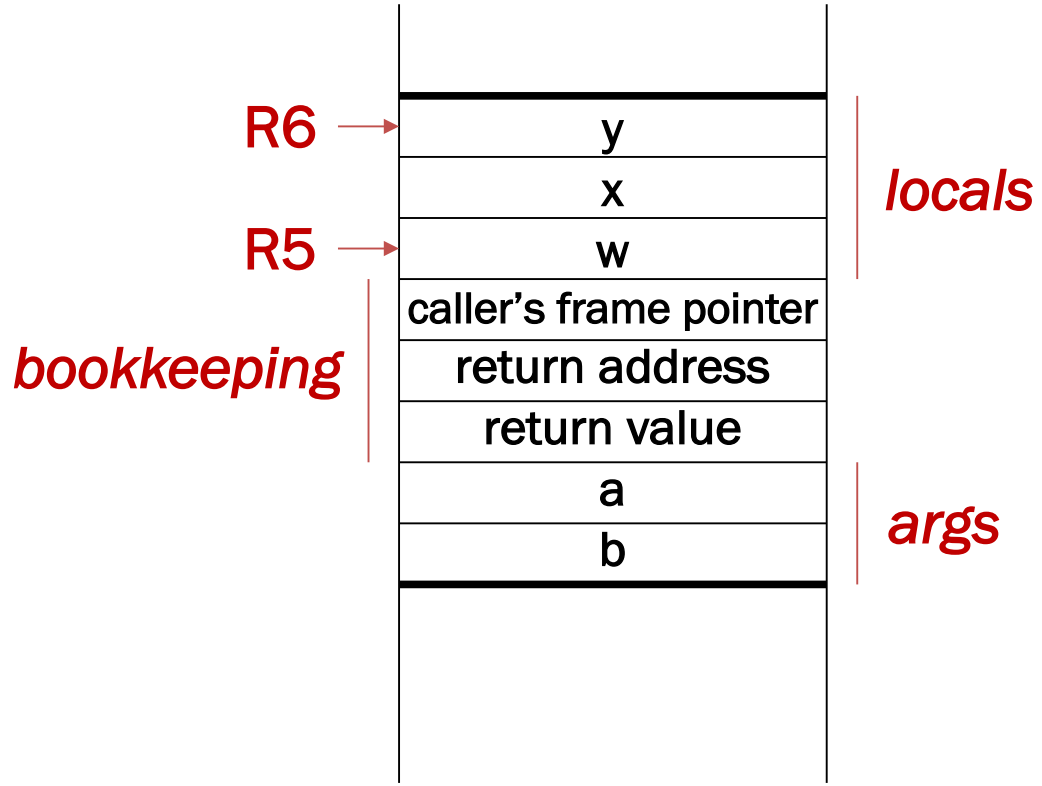
- R5 – **Frame Pointer**. It points to the beginning of a region of activation record that stores local variables for the current function.
- R6 – **Stack Pointer**. It points to the top most occupied location on the stack.
- Arguments are pushed to the stack from **right to left**.
- Local variables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




*Before call* *During call* *After call*

# Activation Record

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



# 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