ECE 220: Computer Systems & Programming

Lecture 7: Functions in C

- MP3 due this Thursday.
- Quiz 1 (02/05 – 02/07)

Midterm 1:

Thursday, 2/15/2024

- Regular Exam: 7:00pm - 8:20pm CT
- ~~Conflict Exam: 5:40pm - 7:00pm CT~~
- **Conflict Exam: Follow the instruction on the course wbsite**
- **Conflict Sign-Up is due by 11:59pm CT on Sunday, 2/11/2024.**

Functions in C

- Functions in C are similar to **subroutines** in LC3 assembly language

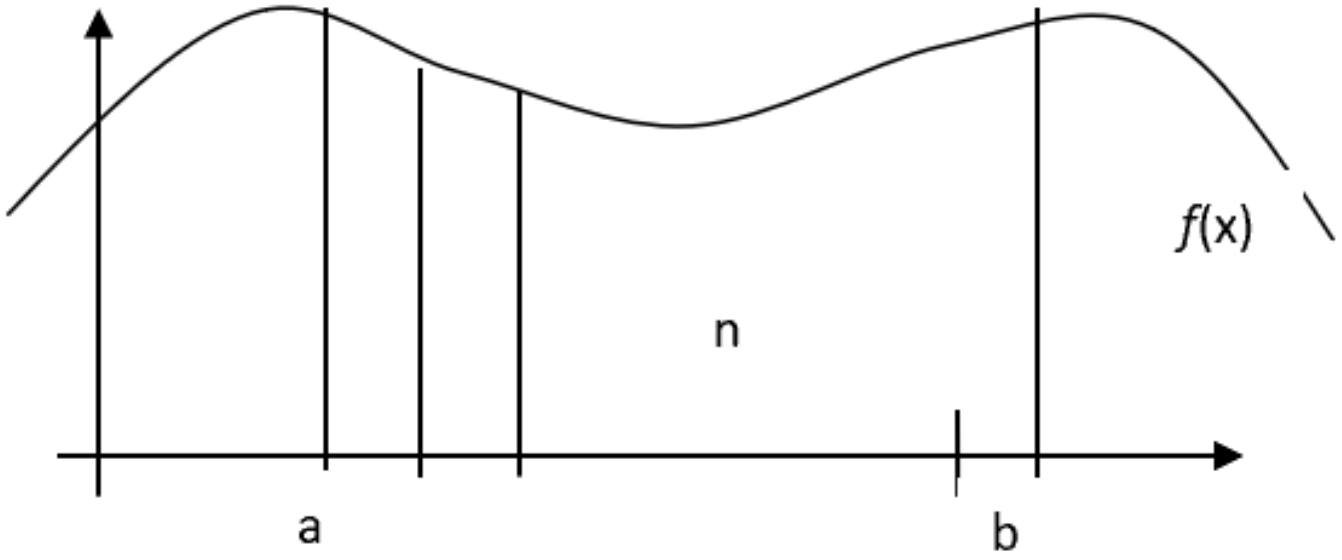
- A piece of code performs certain tasks. Both Provide abstraction

- Using functions enables
  - Hiding low-level details
  - Giving high-level structure to the program that makes it easier to read and understand the flow of the program
  - Enables independent developments (constituent parts in large projects)
  - Efficiently reusing code

Riemann integral

Problem statement: write a program to compute integral of a function f(x) on an interval [a,b].

Algorithm: use integral definition as an area under a function *f*(x) on an interval [a,b]



$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=0}^{n-1} f\left(a + \frac{b-a}{n} i\right) \frac{b-a}{n}$$

```c
/* compute integral of f(x) = x*x+2x+3 on [a,b] */
#include <stdio.h>

int main()
{
    int n = 100;        /* hardcoded number of Reimann sum terms */
    float a = -1.0f;    /* hardcoded [a,b] */
    float b = 1.0f;
    float s = 0.0f;             /* computed integral value */
    int i;                      /* loop counter */
    float x, y;                 /* x and y=f(x) */
    float dx = (b - a) / n;     /* width of rectangles */

    for (i = 0; i < n; i++)
    {
        x = a + dx * i;
        y = x * x + 2 * x + 3;
        s += y * dx;
    }

    printf("%f\n", s);

    return 0;
}
```

Structure of a C function

```c
#include <stdio.h>

float reimann_int(int n, float a, float b);

int main()
{
    int n=100;
    flaot a=-1.0f; float b=1.0f;
    float s;
    s=reimann_int(n,a,b);

    printf("The integral of f(x) is %f\n", s)
}
```

Function Prototype (declaration)

1. Name of function

2. Type of return value

3. Types of parameters

$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=0}^{n-1} f\left(a + \frac{b-a}{n} i\right) \frac{b-a}{n}$$

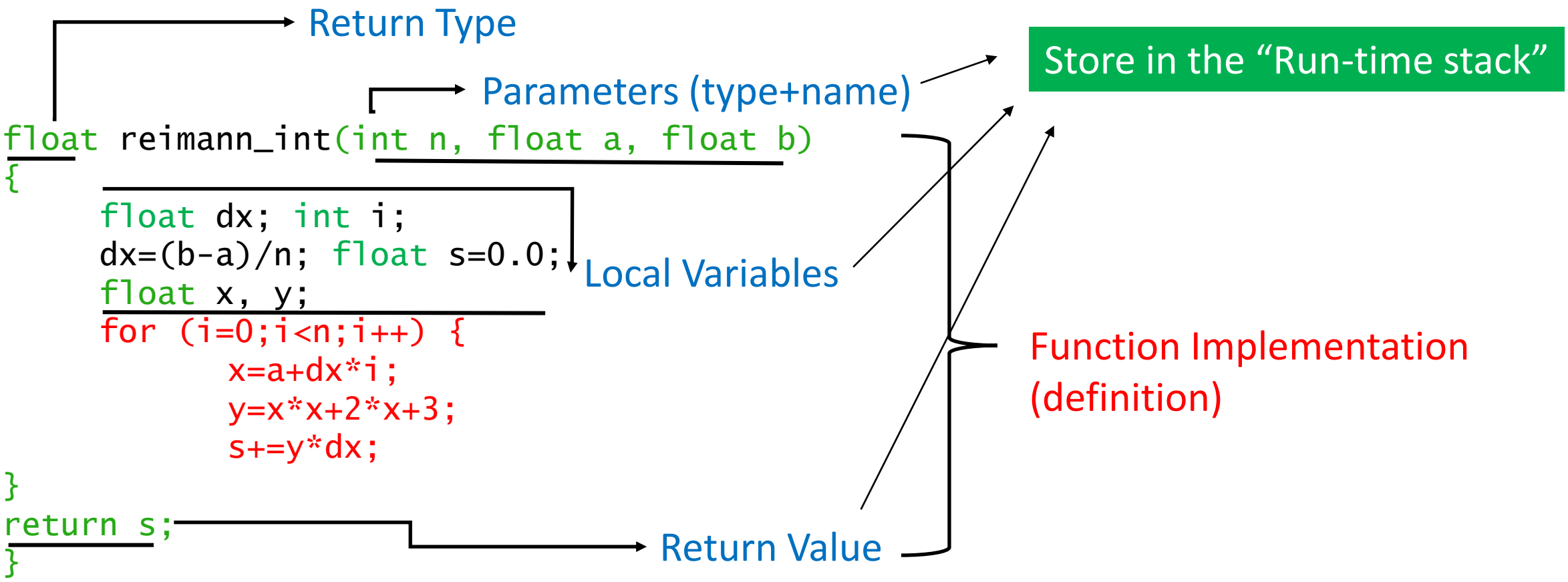
Purpose of function prototype?

→ *informs the compiler about the properties of function*

Function Call

arguments (actual value of parameters)

Structure of a C function (contd.)



Number of Parameters & Return Value

- A function can return at most ONE return value or none
- A function can have multiple parameters or none.

```
int funC(int a, int b){
    return a+b;
}
int funZ(void){
    return 0;
}
void printBanner(){
    printf("============\n");
}
```

→ function call should be `funZ();`

```
int, int funC(int a, int b){
    return a+b, a-b;
}
```

Not Acceptable

Multiple Return Values

Function Can Call Another Function and so on..

```
#include<stdio.h>
float fun(float x);
float reimann_int(int n, float a, float b);
int main()
{
  int n=100;
  float a=-1.0f;
  float b=1.0f;
  float s;
  s=reimann_int(n,a,b);
  printf("The integral of f(x) is %f\n", s);
  return 0;
}
```

```
float fun(float x)
{
        float y;
        y=x*x+2*x+1;
        return y;
}
```

```
float reimann_int(int n, float a, float b)
{
    float dx; int i;
    dx=(b-a)/n; float s=0.0;
    float x, y;
    for (i=0;i<n;i++) {
            x=a+dx*i;
            y=fun(x);
            s+=y*dx;
}
return s;
}
```

Functions can be declared and defined in separate files

main.c

```
#include <stdio.h>
#include "myHeader.h"
int main()
{
  int n=100;
  float a=-1.0f;
  float b=1.0f;
  float s;
  s=reimann_int(n,a,b);
  printf("The integral of f(x) is %f\n", s);
  return 0;
}
```

To compile multiple files use:

```
gcc main.c reimann_func.c
```

myHeader.h

```
float fun(float x);
float reimann_int(int n, float a, float b);
```

Reimann_func.c

```
#include "myHeader.h"
float fun(float x)
{
                float y;
                y=x*x+2*x+3;
                return y;
}

float reimann_int(int n, float a, float b)
{
   float dx; int i;
   dx=(b-a)/n; float s=0.0;
   float x, y;
   for (i=0;i<n;i++) {
                x=a+dx*i;
                y=fun(x);
                s+=y*dx;
}
return s;
}
```

Some Useful Libraries

- stdio.h
  - printf, scanf, getchar, putchar
  - fprintf, fscanf
- math.h
  - cos, sin
  - exp, log, log10, pow
  - ceil, floor, round
  - sqrt
- stdlib.h
  - rand
  - srand
- time.h
  - time(0)

**To compile with math.h,

```
gcc fun_name.c -lm
```

Generate a random number between a and b

```
/* generate 5 pseudo random numbers between a and b */
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int a,b,i;
        a=5;
        b=10;

        for (i=0;i<5;i++)
        {
                printf("%d ",(rand()%(b-a+1)+a));
      /* rand  returns a pseudo-random number in the range of 0 to RAND_MAX. */
        }
printf("\n");

return 0;

}
```

Not so random!

Use srand(n) to generate random numbers

```
/* use srand to generate 5 random number between a and b */
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int a,b,i,n;
        a=5;
        b=10;
        printf("please enter srand seed value: ");
        scanf("%d",&n);
        srand(n);
        for (i=0;i<5;i++)
        {
                printf("%d ",(rand()%(b-a+1)+a));
        }
printf("\n");

return 0;

}
```

Use time(0) to make it true random

```
/* use srand to generate 5 random number between a
and b */
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int main()
{
        int a,b,i;
        a=5;
        b=10;
        srand(time(0));
        for (i=0;i<5;i++)
        {
                printf("%d ",(rand()%(b-a+1)+a));
        }
printf("\n");

return 0;

}
```

A more general implementation of creating an identity matrix and stop printing at any row column position

```
#include<stdio.h>

void create_identity_matrix(int n);
void stop_at_i_j(int n, int i, int j);

int main()
{
   int n,i,j;
   printf("Enter the size of Identity Matrix: ");
   scanf("%d", &n);
   create_identity_matrix(n);
   printf("\n Enter row and column nos. to stop: ");
   scanf("%d%d",&i,&j);
   stop_at_i_j(n, i, j);
   printf("\n");
   return 0;
}
```

```
void stop_at_i_j(int n, int x, int y)
{
            int i,j;

for (i=0;i<n;i++){
            for(j=0;j<n;j++)
            {
          if (i==j)
               printf("1 ");

            else
                        printf("0 ");
            if(i==x && j==y)
                       break;
            if(j==n-1)
                        printf("\n");
            }
if (i==x)
            break;
}
}
```

```
void create_identity_matrix(int n)
{
int i, j;

for (i=0;i<n;i++){
           for(j=0;j<n;j++)
           {
          if (i==j)
                        printf("1 ");
            else
                        printf("0 ");
            if(j==n-1)
                        printf("\n");
           }
if (i==n-1)
           printf("\n");
}
}
```

Implementation of C functions in LC3

- When we compile a C program a Symbol Table is created to keep track of variables
- The symbols tables contains:
  - Variable name (identifier)
  - Variable type
  - Allocated memory location
  - scope

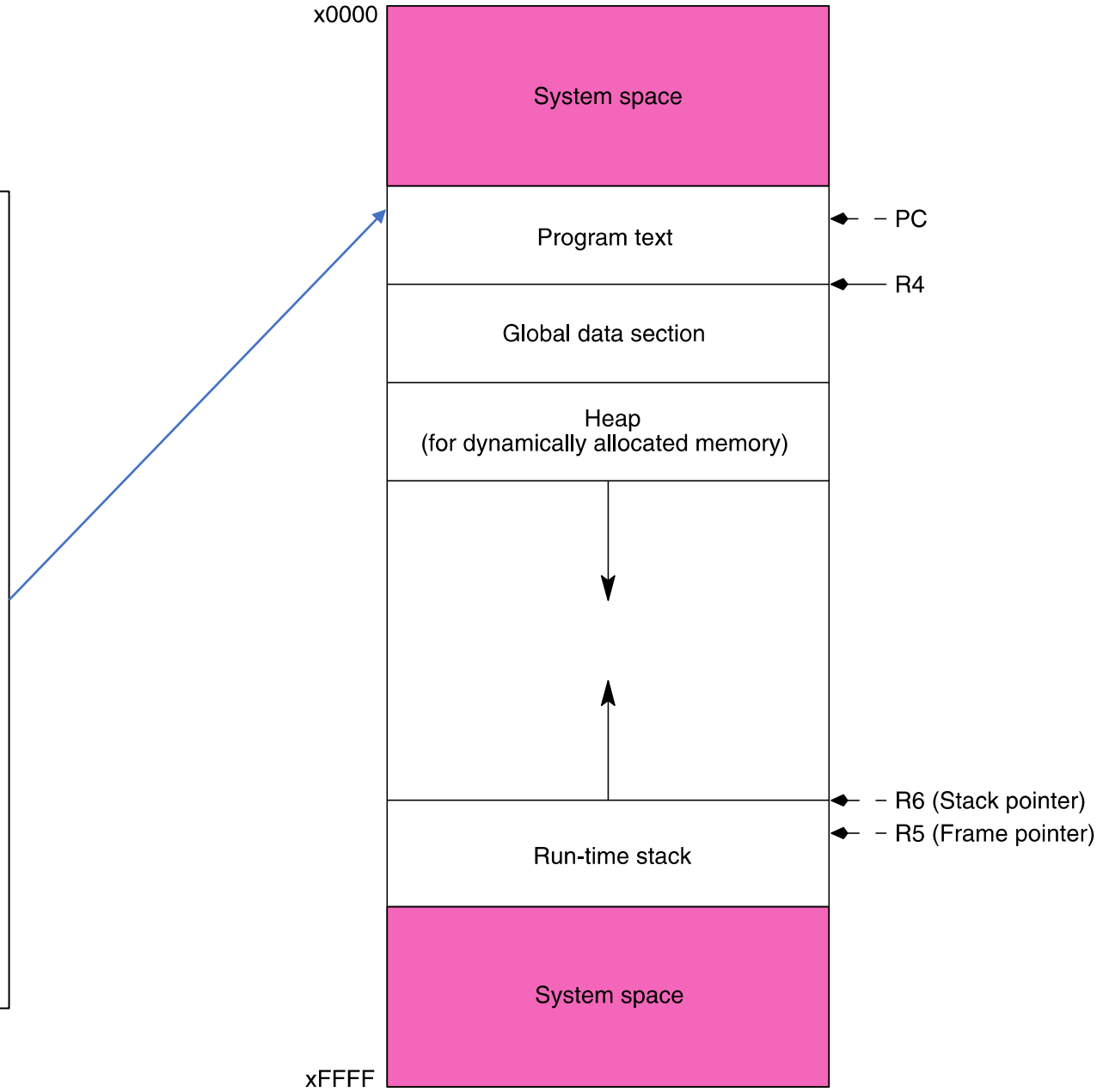
Symbol Table

```
int GlobalX=2;
int GlobalY=4;

int func(int x, int y);

int main()
{
    int x,y,z;
    …

}
int func(int x, int y)
{
    int l,m,n;
    …
}
```

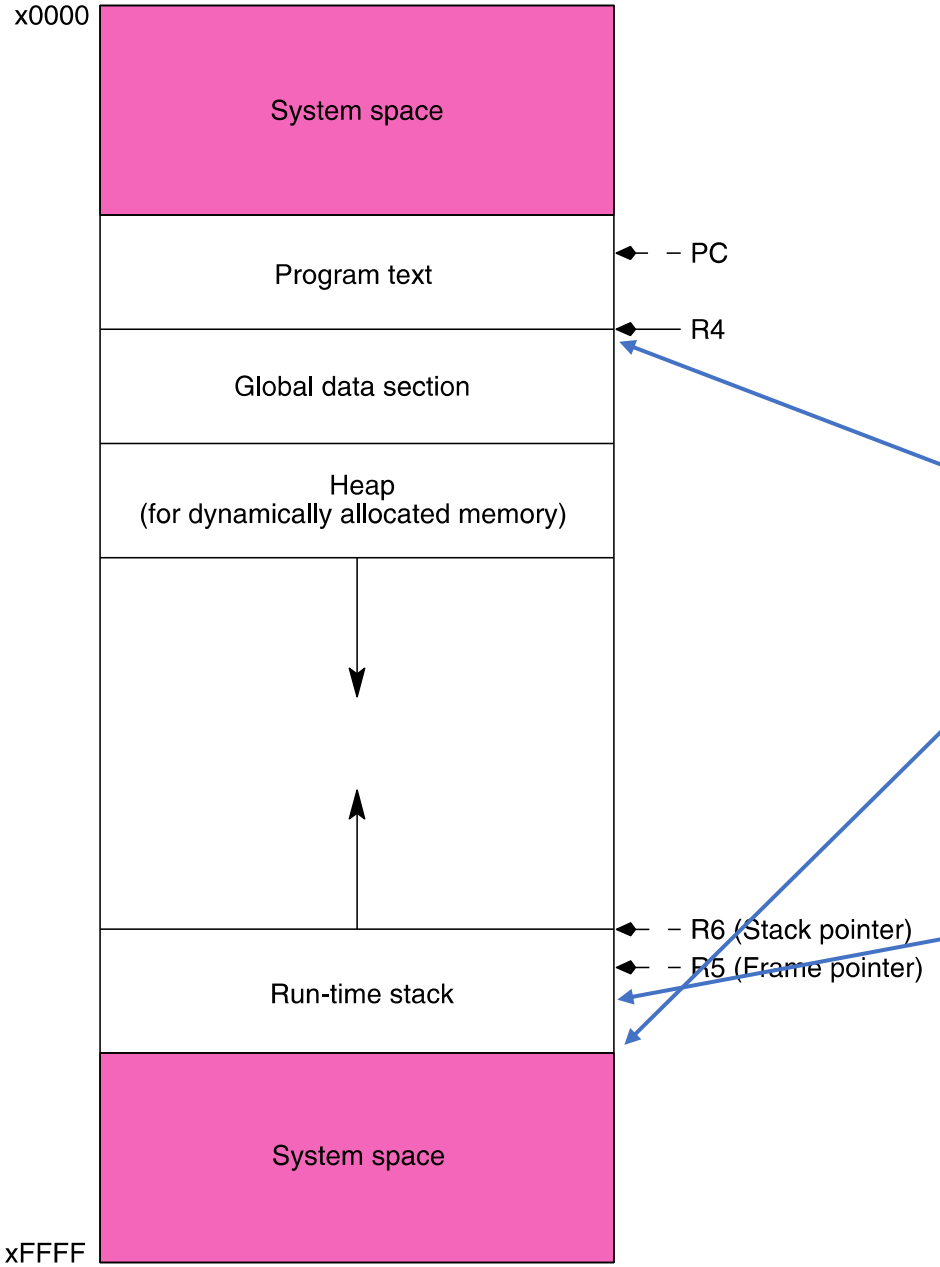
identifier	Type	Location (as an offset)	Scope
GlobalX	int	0	global
GlobalY	int	1	global
x	int	0	main
y	int	-1	main
z	int	-2	main
l	int	0	func
m	int	-1	func
n	int	-2	func

Symbol Table

```
int GlobalX=2;
int GlobalY=4;

int func(int x, int y);

int main()
{
    int x,y,z;
    …

}
int func(int x, int y)
{
    int l,m,n;
    …
}
```



LC3 Memory Map



identifier	Type	Location (as an offset)	Scope
GlobalX	int	0	global
GlobalY	int	1	global
x	int	0	main
y	int	-1	main
z	int	-2	main
l	int	0	func
m	int	-1	func
n	int	-2	func

Global Variables – R4 (Global Pointer)

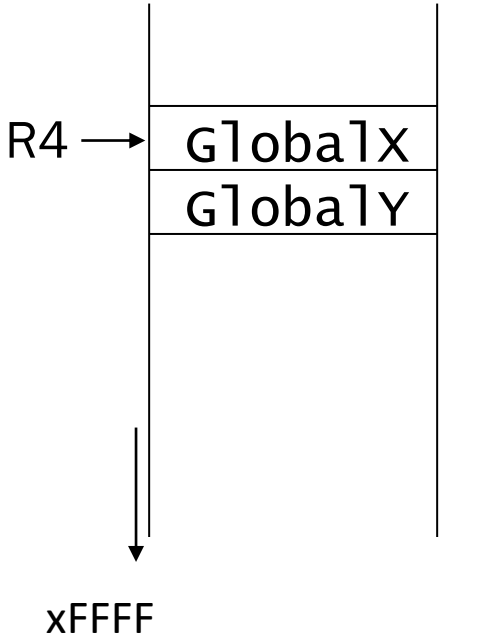
C Code

```
int GlobalX=2;
int GlobalY=4;
```

Symbol Table

Name	Type	Location (as an offset)	Scope
GlobalX	int	0	global
GlobalY	int	1	global

Global data sec



LC-3 Code

```
AND     R0, R0, #0
ADD     R0, R0, #2
STR     R0, R4, #0 ; GlobalX = 2

AND     R0, R0, #0
ADD     R0, R0, #4
STR     R0, R4, #1 ; GlobalY = 4
```

R4 points
the first global variable

Local Variables

C Code

```
int main()
{
    int x,y,z;

    x = 4;
    y = 1;

}
```

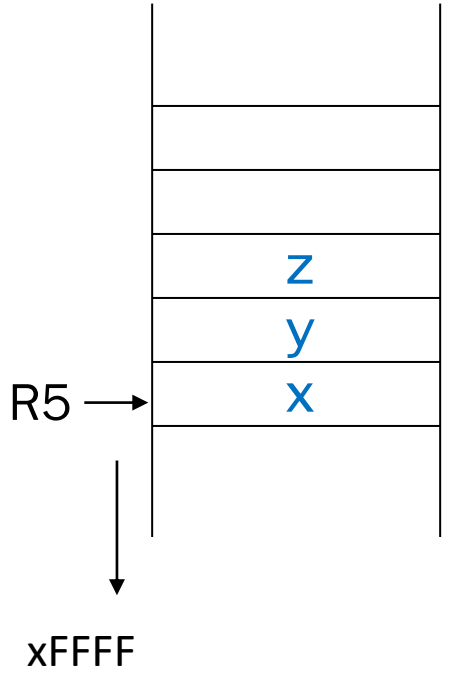
Symbol Table

Name	Type	Location (as an offset)	Scope
x	int	0	main
y	int	-1	main
z	int	-2	main

LC-3 Code

```
AND     R0, R0, #0
ADD     R0, R0, #4
STR     R0, R5, #0 ; x = 4

AND     R0, R0, #0
ADD     R0, R0, #1
STR     R0, R5, #-1 ; y = 1
```

Run-time stack (partial)



R5 points the first local variable

Local Variables in Activation Record

- Every *function call* creates an *activation record (or stack frame)* and *pushes* it onto the run-time stack.
- Local variables are one part of the activation record.
- Whenever a function *completes (return)*, the activation record is *popped* off the run-time stack.
- Whenever a function calls another one (*nested*), the run time stack *grows* (push another activation record onto the run-time stack).

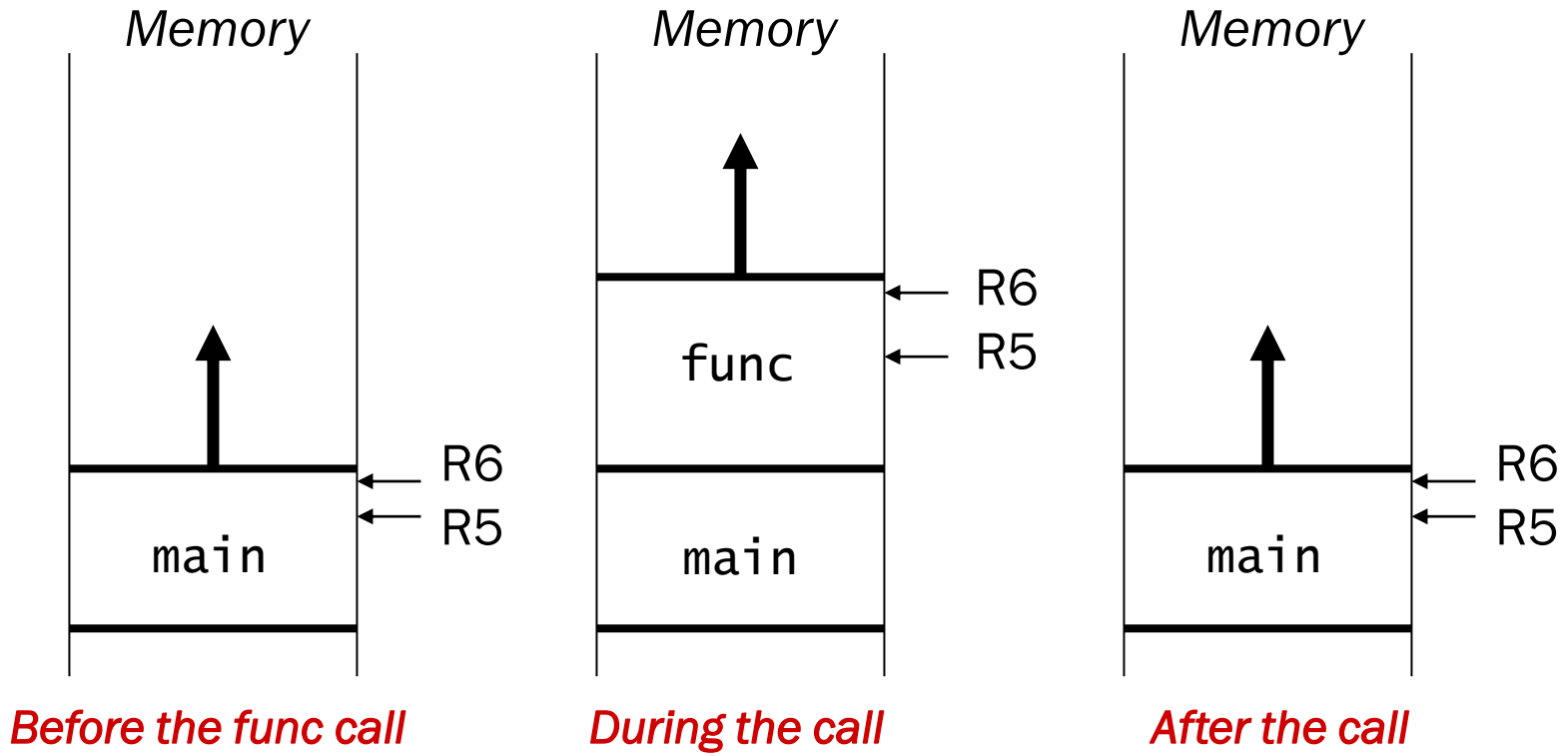
one function call = one activation record

A function could call **itself** (recursion)

Activation Record

- stored in run-time stack
- *function call = push activation record*
- *function return = pop activation record*

```
In func(int x, int y);

int main()
{
    int x,y,z;
    x = func(5, 7);

}
```

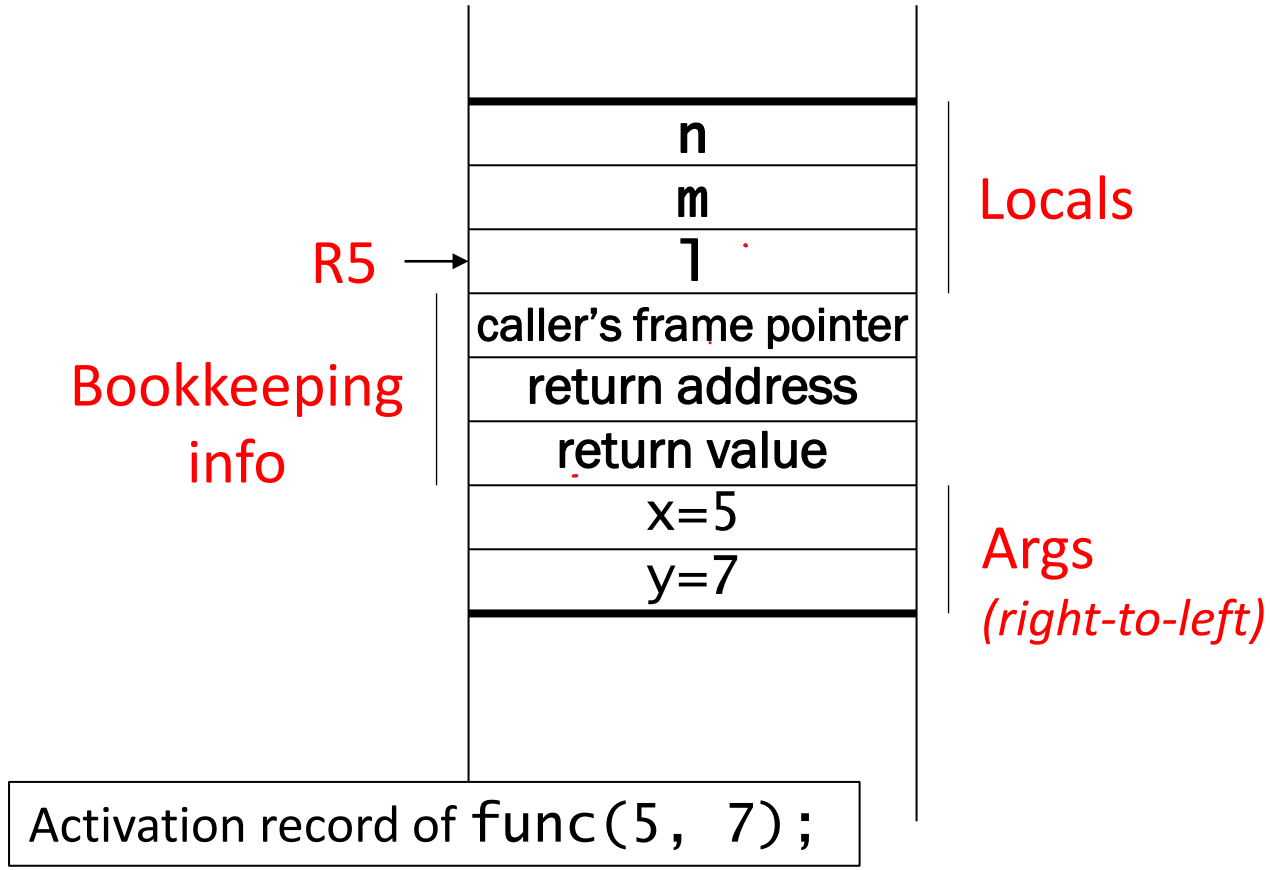


Building Activation Record

- Information about each function call, including

1. Arguments *2. Bookkeeping info* *3. Local variables*

```
int main()
{
    int x,y,z;
    x = func(5, 7);

}
int func(int x, int y)
{
    int l,m,n;
    .
    .
    .
    return l;
}
```



Midterm1 Review:

- Basic concept on LC-3
  - memory, processing unit, control unit, input/output
- Memory mapped I/O
  - KBDR, KBSR, DDR, DSR
  - Basic input/output routine by polling
- TRAP
  - TRAP mechanism operation (TVT, TRAP service routine, …)
- Subroutine
  - How to write a subroutine (callee/caller-save, RET, R7, nested subroutine, …)
- Stack
  - PUSH, POP, TOS(R6)
- **Basics of C** (First two lectures on C)
- **Review your MP1, MP2, MP3, and worksheets**
- Be familiar with LC-3 instructions (e.g. ST/LD family, Branch)