Lecture 7:

Functions: Sub Programs

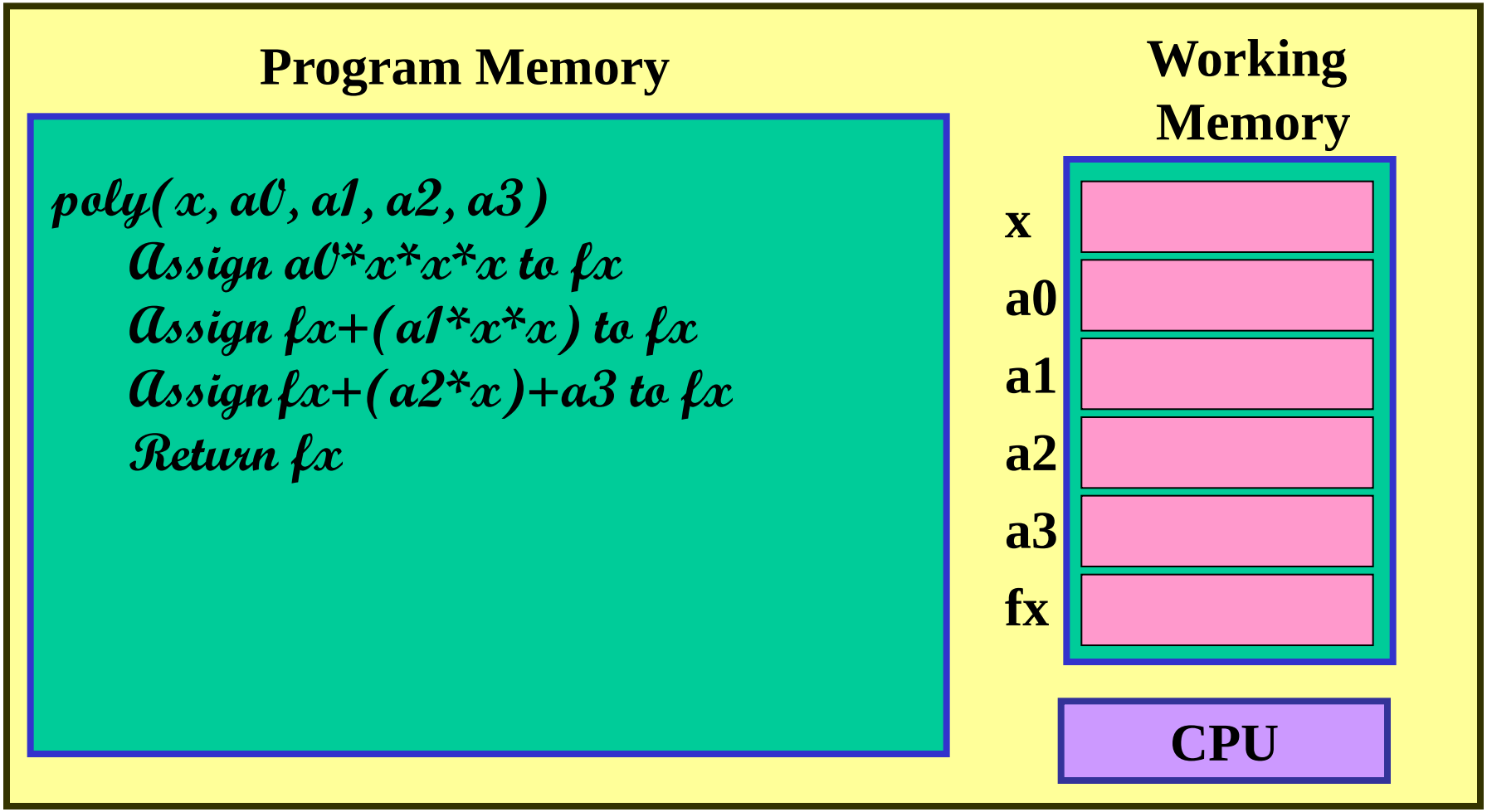
Prof. Shervin Shirmohammadi
University of Ottawa

Subprograms and C Functions

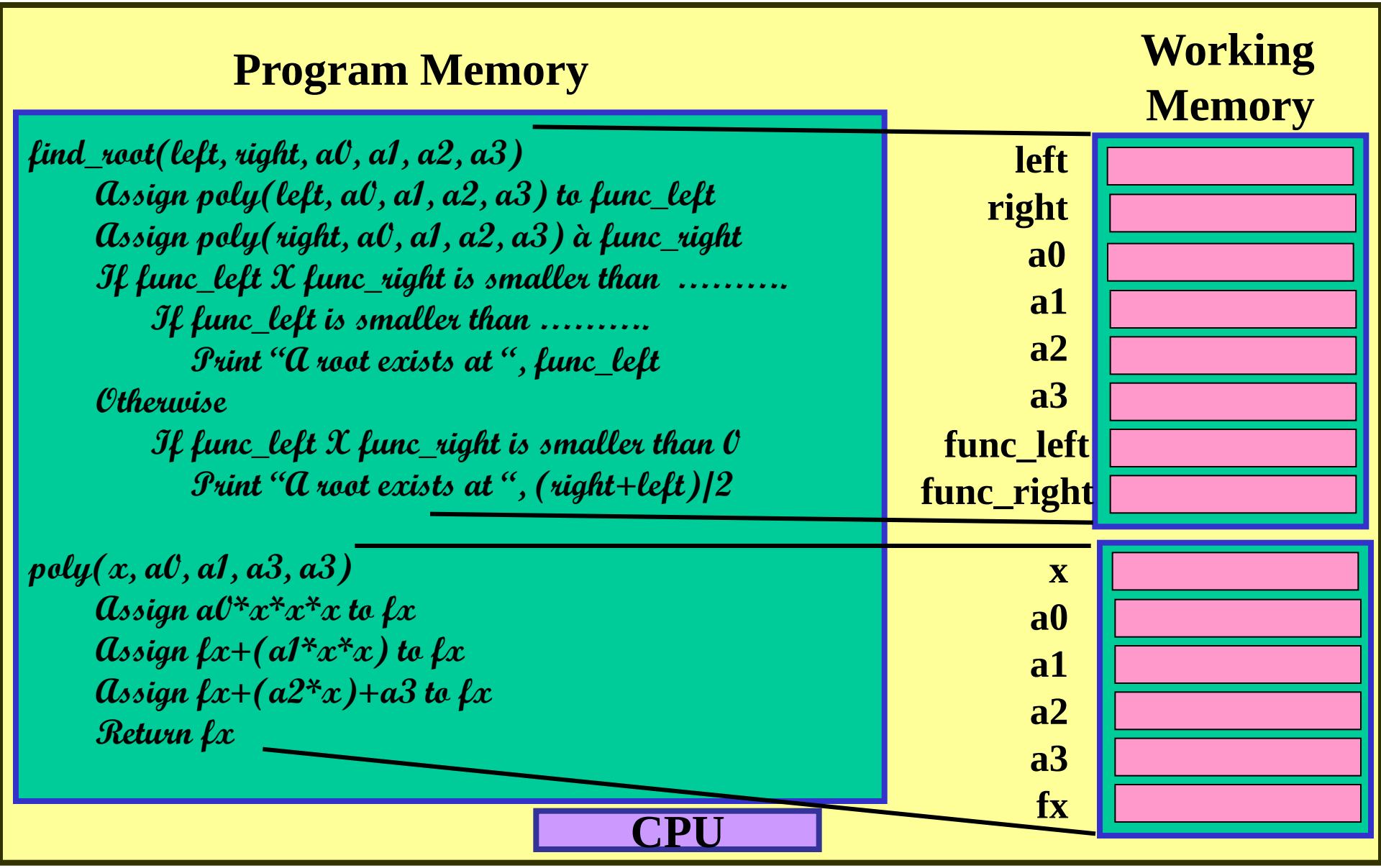
- A large program is easier to code, manage and maintain if it is written in a modular fashion.
  - Modular program design means that specific tasks are isolated and coded into separate modules or subprograms.
  - In C, a module or subprogram is called a function.
- Modular program design using functions offers many advantages since functions:
  - allow the abstraction of code during program design and development,
  - allow the re-use of code elsewhere in the program or in another program,
  - avoid the repetition of code in the same program or in another program.

The Subprogram

- The subprogram is composed of a name, parameters and an instruction bloc; and it may return a value
  - The parameters are variables used to receive values from the caller



Calling a subprogram



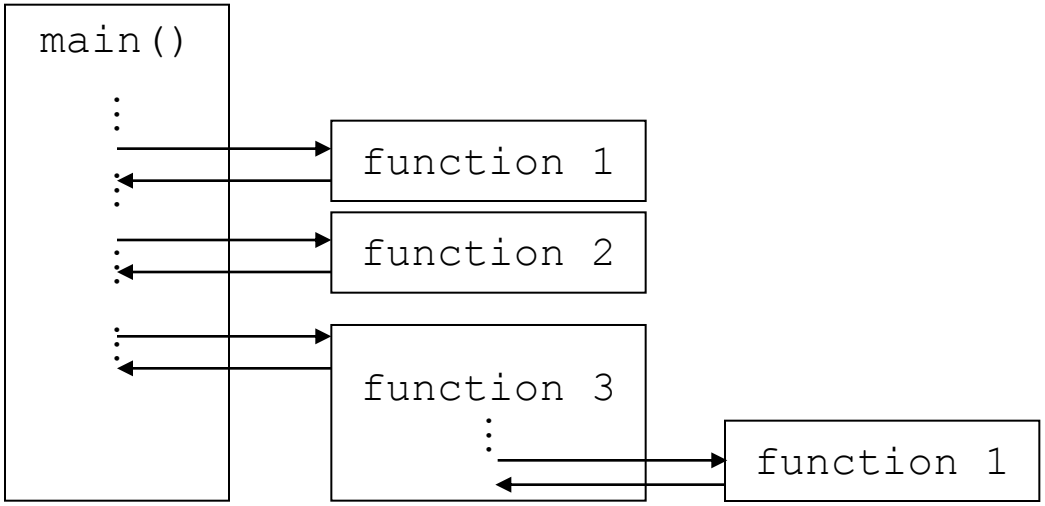
Subprogram operation

- A piece of working memory is reserved for and during the execution of a subprogram.
- The parameters and variables are created in the reserved working memory when the subprogram is executed.
- Note that the same name for variables can be used in different subprograms:
  - Although the names are the same, the variables are different – why?
  - The variables (and parameters) are **local** to the subprogram.
- When the execution of the subprogram is completed, the variables (and paramters) no longer exist and the working memory can be re-used (e.g. for the execution of another subprogram)

Calling a subprogram

- A call to a subprogram can be made within an instruction using the subprogram name and a set of arguments
  - When a subprogram is called, the execution of the calling subprogram is suspended
  - The arguments are expressions that are evaluated to give values
  - The argument values are stored in the parameters of the subprogram to be executed
    - The order of the arguments correspond to the order of the parameters
  - The subprogram can return values to the calling subprogram (or calling main program)
    - This means that a subprogram call can be used in any expression just like a variable
    - E.g. ***Assign 3+poly(4.2, 3.2, -2.5, 1, 8.8) to z***

- Execution flow with many function calls:



- Functions can also return a value back to the caller.

Definition of Functions

- It is possible to program your own functions in C; this is called function definition. The syntax of a function definition in C is:

```
type function_name(parameter list)
{
    C declarations;
    C commands;
}
```

Careful: there is no ; here.

  - The function header describes the communication channel that will be established with the caller.
  - The name of the function is created from the same symbols as a variable name.
  - The type of the function identifies the type of the value returned by the function.
    - If type is `void` then the function returns nothing to the caller.
    - If type is omitted then the compiler assumes that the function returns an `int`.

Function Parameters and Variables

- The parameter list contains the declaration of the variables to receive arguments passed to the function.
  - Individual declarations are separated by `,` (commas).
  - If the function does not receive arguments then the parameter list is of type `void`.
  - If the type of a parameter is omitted then the compiler assumes that it is an `int`.
- All variable declarations and instructions in the function are placed between `{}`; i.e., an instruction bloc.
  - All variables declared in a function are local; this means that they are not available and are not known outside of the function where they are defined.

Returning from a Function

- There are 3 ways of returning program control from a function back to the caller:
    1. Reaching the end of the function as delimited by the `}`
    - 2. By executing the command: `return;`
    - 3. By executing the command: `return expression;`
  - Mechanisms 1 or 2 are used in a function that does not return a value to the caller; **i.e: functions of type `void`.**
  - Mechanism 3 must be used in a function that returns a non-void value to the caller; `expression` must evaluate to a value having the same type as the function declaration.
- A function cannot be defined within another function.

- Example: a simple function that prints out Hello World! to the screen.

```
void hello(void)
{
   printf("Hello World!\n");
}
```

This function receives no arguments and returns nothing to the caller.

- Example: a simple function that prints out the largest of 2 integers to the screen.

```
void prtmax(int a, int b)
{
    if (a > b)
       printf("%d is the largest integer.\n", a);
    else
       printf("%d is the largest integer.\n", b);
}
```

This function receives 2 arguments (integers) and returns nothing to the caller.

- Example: a simple function that returns the largest of 2 integers to the caller.

```
int max(int a, int b)
{
    if (a > b)
       return a;
    else
       return b;
}
```

This function receives 2 arguments (integers) and returns an integer to the caller.

- Example: a simple function that returns the largest of 2 integers to the caller.
  - Performs the same task as the previous example but in better style.

```
int max(int a, int b)
{
   int largest;

   if (a > b)
      largest = a;
   else
      largest = b;

   return largest;
}
```

This function receives 2 arguments (integers) and returns an integer to the caller.

Function Prototypes

Careful: there is a ; here.

- The syntax of a function prototype is:
  - `type function_name(type, type, … , type);`
- The prototype:
  - declares that `main()` will call the function: `function_name`,
  - defines the type of the value returned by the function,
  - identifies the number of parameters and the order of the types passed to the function.
- The prototype of a function must have:
  - the same name as in the function definition,
  - the same function type as in the definition,
  - the same number of parameters as in the definition, and
  - the same order of parameter types.
  - The parameter names are optional.
- The function prototypes are usually listed after the preprocessor directives but before `void main()` in the program file.

Function Prototypes

- The prototypes for our simple example functions are:
  - `void hello(void);`
  - `void prtmax(int, int);`
  - `int max(int, int);`
- Heading files such as `stdio.h`, `math.h` and `stdlib.h` contain among other things, function prototypes for the standard C functions.
- Function prototypes are not required in a C program but you are strongly encouraged to include them for all functions called in the main program since they:
  - help others understand your program file,
  - can help avoid errors when calling the function.

Put it All together!

```
#include <stdio.h>

int add2nums( int firstnum, int secondnum );

int main(void) {
  int y,a,b;

  printf("Enter 2 numbers\n");
  scanf("%d%d", &a, &b);

  y = add2nums(a,b);

  printf("a is %d\n", a);
  printf("b is %d\n", b);
  printf("y is %d\n", y);
  return(0);
}

int add2nums( int firstnum, int secondnum ) {
  int sum;
  sum = firstnum + secondnum;
  return(sum);
}
```

Function Prototype

Function call

Function definition

Example

- Write a function that accepts three float numbers and returns the average
- Write the main() function that prompts the user for 3 numbers, call the function, receive the result, and prints the result on the output screen.