Lecture 9: Program Organization

Sarah Nadi
nadi@ualberta.ca
Department of Computing Science
University of Alberta

CMPUT 201 - Practical Programming Methodology
Winter 2018

[With material/slides from Guohui Lin, Davood Rafei, and Michael Buro. Most examples taken from K.N. King's book]

Agenda

- Local variables
- External variables
- Blocks
- Scope
- Organizing a C program

Readings

- Textbook Chapter 10

Organizing Functions

- Last week, we learned about functions
- Today, we will learn how to organize our program when we have multiple functions
  - How do we differentiate the variables in our functions and in our program?
  - Where can these variables be accessed or modified?
  - Can we re-use variable names in our program?

Variable “Dimensions”

- *Scope*
  - the part of the program in which this variable can be referenced
  - two types of scopes: *block scope* and *file scope*
- *Storage duration*
  - the portion of a program execution during which storage for the variable exists
  - two types of storage duration: *automatic storage duration* and *static storage duration*

Local Variables

- Local variables are declared in the body of a function and are local to the function

- They have a block scope: are visible only from the declaration of the variable to the end of the smallest enclosing body

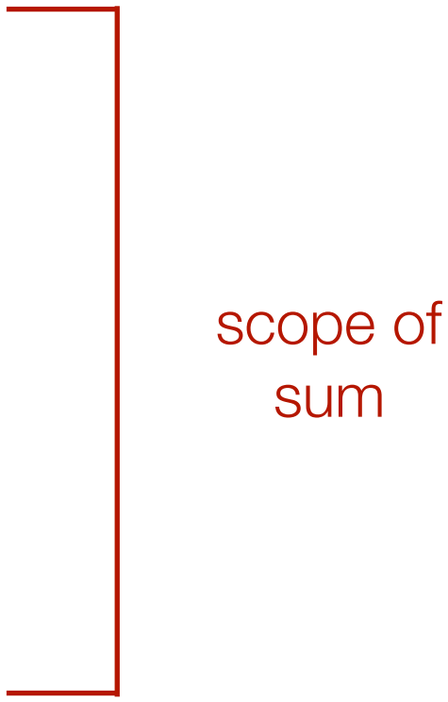
- They have automatic storage duration: life span/storage duration is the same as that of the function

Local Variables Example

```
void sum_avg (int numbers[], int n){
  int sum = 0;

  for(int i = 0; i < n; i++){
    sum += numbers[i]
  }

  int avg;
  avg = sum/n;
  printf("sum = %d\n", sum);
  printf("average=%d\n", avg);
}
```

Local Variables Example

```
void sum_avg (int numbers[], int n){
  int sum = 0;

  for(int i = 0; i < n; i++){
    sum += numbers[i]
  }

  int avg;
  avg = sum/n;
  printf("sum = %d\n", sum);
  printf("average=%d\n", avg);
}
```



Local Variables Example

```
void sum_avg (int numbers[], int n){
  int sum = 0;

  for(int i = 0; i < n; i++){
    sum += numbers[i]
  }

  int avg;
  avg = sum/n;
  printf("sum = %d\n", sum);
  printf("average=%d\n", avg);
}
```

scope of sum

scope of avg

Local Variables Example

```
void sum_avg (int numbers[], int n){
  int sum = 0;

  for(int i = 0; i < n; i++){
    sum += numbers[i]
  }

  int avg;
  avg = sum/n;
  printf("sum = %d\n", sum);
  printf("average=%d\n", avg);
}
```

scope of i

scope of avg

scope of sum

Local Variables Example

```
void sum_avg (int numbers[], int n){
  int sum = 0;

  for(int i = 0; i < n; i++){
    sum += numbers[i]
  }

  int avg;
  avg = sum/n;
  printf("sum = %d\n", sum);
  printf("average=%d\n", avg);
}
```

scope of i

scope of avg

scope of sum

- `sum`, `avg`, and `i` have automatic storage duration.

Local Variables Example

```
void sum_avg (int numbers[], int n){
  int sum = 0;

  for(int i = 0; i < n; i++){
    sum += numbers[i]
  }

  int avg;
  avg = sum/n;
  printf("sum = %d\n", sum);
  printf("average=%d\n", avg);
}
```

scope of i

scope of avg

scope of sum

- `sum`, `avg`, and `i` have automatic storage duration.
- The storage of `sum` and `avg` is automatically allocated when the enclosing function is called and deallocated when the function returns (i.e., they cease to exist beyond the function).

Local Variables Example

```
void sum_avg (int numbers[], int n){
  int sum = 0;

  for(int i = 0; i < n; i++){
    sum += numbers[i]
  }

  int avg;
  avg = sum/n;
  printf("sum = %d\n", sum);
  printf("average=%d\n", avg);
}
```

scope of i

scope of avg

scope of sum

- `sum`, `avg`, and `i` have automatic storage duration.
- The storage of `sum` and `avg` is automatically allocated when the enclosing function is called and deallocated when the function returns (i.e., they cease to exist beyond the function).
- The storage of `i` is automatically allocated when the for loop is entered and is deallocated when the loop is exited.

Local Variables Example

```
void sum_avg (int numbers[], int n){
  int sum = 0;

  for(int i = 0; i < n; i++){
    sum += numbers[i]
  }

  int avg;
  avg = sum/n;
  printf("sum = %d\n", sum);
  printf("average=%d\n", avg);
}
```

scope of i

scope of avg

scope of sum

- `sum`, `avg`, and `i` have automatic storage duration.
- The storage of `sum` and `avg` is automatically allocated when the enclosing function is called and deallocated when the function returns (i.e., they cease to exist beyond the function).
- The storage of `i` is automatically allocated when the for loop is entered and is deallocated when the loop is exited.
- Every time `sum_avg` is called, new storage/memory is allocated to `sum` and its value is initialized to 0.

Blocks

```
{
 declarations
 statements
}
```

- a block is enclosed in parentheses

- it acts like a function and has its on local variables

- any variables declared within this block are local to the block, and cannot be referenced from outside the block

Function Parameters

- are similar to local variables
- parameters are local to the function
- have automatic storage duration (life span the same as the function)
- block scope (they are visible only in the function)

Static Local Variables

- Declared in the body of the function using the keyword `static`
- They are still local to the function and have block scope, BUT
- They have permanent storage duration (life span the same as the entire program)
- This means that static local variables retain their value throughout the execution of the program (i.e., they can keep their value even after execution leaves the function)
- Static local variables provide a place to hid data (from other functions) for future calls of the same function/block

demo: static.c

External/Global Variables

- We saw global variables in the first class
- Global variables are useful when multiple functions need to access/modify the same variable
- However, it is generally safer to pass information through function parameters
- Global variables are declared outside the body of a function
  - they have static storage duration (similar to declaring a variable as static)
  - they have file scope (visible from their declaration to the end of the closing file)

External/Global Variables

Cont'd

- Pros
  - convenient: no need to worry about using parameters
- Cons
  - changing the variable type could be problematic
  - difficult to locate errors
  - hard to reuse the same functions
  - may lead to unexpected name conflicts:

```
int i;

void f(void){
  int i;
  for (i = 0; i <n; i++)
  …
  }
}
```

Using External Variables Implement a Stack

- What is a stack?
  - a data structure that is similar to an array but you cannot access elements by index/position
  - Instead, you can
    - *push*: add an element to the top of the stack
    - *pop*: remove an element from the top of the stack
  - a stack usually has a variable `top` to indicate the number of elements on the stack
- Possible implementation
  - can use an array for the stack
  - can have two separate functions, one for each operation
  - can have the array and top as external/global variables

demo: stack.c

Scope

- The same identifier may be used in different places to mean different things
- When a declaration inside a block names an identifier that's already visible, the new declaration temporarily "hides" the old one, and the identifier takes on a new meaning

```
int i ;                 /* Declaration 1 */

void f(int i )          /* Declaration 2 */
{
  i = 1;
}

void g(void)
{
  int i = 2;            /* Declaration 3 */

  if (i > 0) {
    int i ;             /* Declaration 4 */

    i = 3;
  }

  i = 4;
}

void h(void)
{
  i = 5;
}
```

Organizing a C Program

demo: example/

Organizing a C Program

- Separate function prototypes into a header file (.h)

demo: example/

Organizing a C Program

- Separate function prototypes into a header file (.h)
- Group related functionality into a single .c file (with the corresponding .h file) and separate the files in your program (e.g., stack.h, cipher.h, list.h)

demo: example/

Organizing a C Program

- Separate function prototypes into a header file (.h)
- Group related functionality into a single .c file (with the corresponding .h file) and separate the files in your program (e.g., stack.h, cipher.h, list.h)
- Some rules:
  - a directive does not take effect until the lines it is defined on
  - a type name cannot be used until it is defined
  - a variable cannot be used until it's declared (but will have garbage value if not properly initialized)
  - a function needs to be declared or defined before it is called

demo: example/

Recall: Compiling a C Program (Behind the Scenes)