CH-230-A

Programming in C and C++

C/C++

Lecture 4

Dr. Kinga Lipskoch

Fall 2022

Local Variables

- Variables can be declared inside any function
    - These are called local variables
    - Local variables are created when the function is called (e.g., the control is transferred to the function) and are destroyed when the function terminates
- Local variables do not retain their values between different calls

The Concept of Scope

▶ The scope of a name (function, variable, constant) is the part of the program where that name can be used

▶ The scope of a local variable is the function where it is defined
  ▶ From the point of its definition

▶ Names having different scopes do not clash

Global Scope

▶ The scope of the names of functions goes from the prototype/definition to the end of file

▶ After their name is known they can be used, i.e., called

▶ It is possible to define global variables, i.e., variables outside function

    ▶ Their scope is from the point of definition to the end of the file

    ▶ After their definition is given they can be used, i.e., written and read

Local and Global Scope

```c
1  #include <stdio.h>
2
3  //global variable
4  int x = 7;
5
6  void xlocal(int y)  {
7    int x;
8    x = y * y;
9    printf("xlocal: %d\n", x);
10   return;
11 }
12
13 void xglobal(int y)  {
14   x = y * x;
15   printf("xglobal: %d\n", x);
16   return;
17 }
```

```c
1  int main() {
2    //int x;
3    // try to explain if not
4    // commented out
5    x = 8;
6    printf("main: %d\n", x);
7    xlocal(x);
8    printf("main: %d\n", x);
9    xglobal(x);
10   printf("main: %d\n", x);
11   return 0;
12 }
```

Do not Misuse Global Variables

- ▶ Global variables can be used to communicate parameters between functions
- ▶ They can introduce subtle bugs in your code
- ▶ In general try to avoid them unless enormous advantages can be gained at a price of low risk
  - ▶ Document why you insert them
- ▶ Bigger projects will avoid using global variables

Parameters

▶ Function parameters are treated as local variables

▶ Local variables within functions and parameters must have different names

▶ Therefore the scope of a parameter is its function

Parameters: by Value and by Reference

- ▶ **By value**: variables are copied to parameters
  - ▶ Changes made to parameters are not seen outside the function
- ▶ **By reference**: variables and parameters coincide
  - ▶ Changes made to parameters are seen outside the function
  - ▶ In C this is obtained by mean of pointers

Example: Passing by Value (1)

```c
#include <stdio.h>
void increase(int par) {
  par++;
}
/*  In this case no prototype:
    can you tell why? */
int main() {
  int number = 5;
  increase(number);
  printf("Increased number is %d\n", number);
  /* not as expected? */
  return 0;
}
```

Example: Passing by Value (2)

1) | 5 |
    number

2) | 5 |
    par

3) | 6 |
    par

4) | 6 |
    par

5) | 5 |
    number

Parameters by Reference in C

- ▶ C passes only parameters by value
- ▶ For references it is necessary to provide a pointer to the variable
- ▶ In order to make a modification visible
- ▶ Outside it is necessary to use the dereference (∗) operator

Example: Passing by Reference (1)

```c
#include <stdio.h>

void increase(int *par) {
  *par = *par + 1;
}

int main() {
  int number = 5;
  increase(&number); /* pass pointer */
  printf("Increased number is %d", number);
  return 0;
}
```

Example: Passing by Reference (1)

1) **5**

   number

2) **5**

   par is pointing to number par = &number
   par is the copy of the memory address of number

3) **6**

   number manipulated via pointer par

4) **par is deleted as the copy of the address**

5) **6**

   number

Indentation Styles (1)

- ▶ Use spaces between operators: `a = b + 5;`
- ▶ Exception: `b++;`
- ▶ Do not use spaces if parentheses act as delimiter (functions)
  `printf("Number %d", b);`
- ▶ But use spaces before after `if`, `for`, `while`:
  `while (i <= 10)`
- ▶ Always put a space after comma
- ▶ Do not put a space before semicolon:
  `printf("Number %d", b);`

Indentation Styles (2)

▶ Put the opening brace either behind last word (including space) or put it on the next line

▶ Indent the block inside by tab or 4 (8) spaces

▶ The closing brace should be on the same column as the opening statement

```
1 for (i = 0; i < 10; i++) {    // K&R style
2   printf("%d\n", i);
3 }
```

or

```
1 for (i = 0; i < 10; i++)      // Allman style
2 {
3   printf("%d\n", i);
4 }
```

Strings

- ▶ A string is a sequence of characters
- ▶ Strings are often the main way used to communicate information to the user
- ▶ Many languages provide a string data type, but C does not
- ▶ In C strings are treated as arrays of characters
- ▶ `char my_string[30];`

C Strings

- ▶ A string is represented as a sequence of chars enclosed by double quotes
    - ▶ "This is it"
- ▶ String are stored in arrays of chars
    - ▶ An extra character is always added at the end to mark the end of the string
    - ▶ The extra character is the '\0' character i.e., the character whose ASCII code is 0

T	h	i	s		i	s		i	t	\0

fgets versus gets (1)

- ▶ gets does not check if you type more characters than allowed:
  ```
  char inputString[50];
  gets(inputString);
  ```
- ▶ fgets allows additional parameters:
  ```
  char line[50];
  fgets(line, sizeof(line), stdin);
  ```
  - ▶ Reads up to 49 characters from the input stream
  - ▶ The 50th one is used to store the null character '\0'

fgets versus gets (2)

- ▶ gets replaces the trailing '\n' with a '\0'
- ▶ fgets does not replace '\n', but it leaves it in the string
- ▶ Read the man pages for learning more on these functions
    - ▶ man gets
    - ▶ man fgets
- ▶ To make your life easier use fgets and convert to integer via sscanf
- ▶ Avoid using gets, it is unsafe

fgets and scanf together

▶ scanf and fgets do not work well together

▶ Your code should look like this, if you use both

```
1    scanf("%d", &number);
2    getchar();
3    ...
4    fgets(line, sizeof(line), stdin);
5    sscanf(line, "%d", &number);
```

String Functions

- ▶ Defined in string.h
- ▶ strlen   Determines the length of a string
- ▶ strcat   Concatenates two strings
- ▶ strcpy   Copies one string into another
- ▶ strcmp   Compares two strings
- ▶ strchr   Searches a char in a string
- ▶ See man pages
  - ▶ Do not reinvent the wheel, there are many many functions that will help you

Passing Arrays to Functions

▶ An array does not store its size

▶ This has to be provided as a parameter, or by making assumptions on the contents of the array (like for strings)

▶ The name of an array is a pointer to the first element of the array, i.e., when an array is passed to a function, a copy of the address of the first element is given

▶ Modifications to the elements are seen outside

▶ Modifications to the array are not seen outside

▶ Can you explain why?

Passing Arrays to Functions: Example

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  void strange_function(int v[], int dim) {
4    int i;
5    for (i = 0; i < dim; i++)
6      v[i] = 287;
7    // v = (int *) malloc(sizeof(int) * 1000);
8  }
9  int main() {
10   int array[] = {1, 2, 9, 16};
11   int *p = &array[0];
12   strange_function(array, 4);
13   printf("%d %p %p\n", array[0], p, array);
14   return 0;
15 }
```