CH-230-A

Programming in C and C++

C/C++

Lecture 2

Dr. Kinga Lipskoch

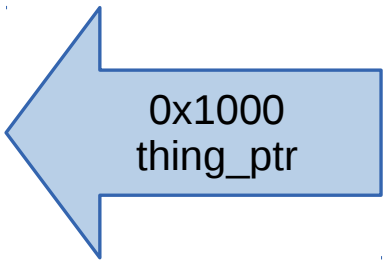
Fall 2020

Pointers: Introduction (1)

- Every data is stored in memory
- The memory is organized as a sequence of increasingly numbered cells
  - The number of a cell is its address
  - The address is determined when the variable is declared

0x1000

A thing



A pointer

Pointer Comparison

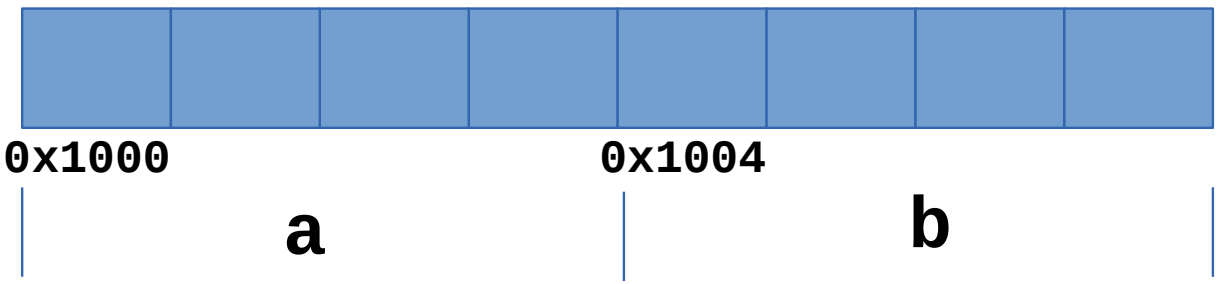
- You can compare pointers and data to street addresses and houses
- Houses may vary in size, while the address which points to the house is approximately of the same size and small
- More than one pointer may point to the same thing

Pointers: Introduction (2)

- Consider the following declaration

```
int a = 145, b = 98;
```

- Supposing that an `int` occupies 4 bytes the following situation could arise



The address of `a` is `0x1000` and the address of `b` is `0x1004`.

Pointers

- A pointer is a variable which stores the address of another variable
- In the previous example, a pointer to `a` would store `0x1000`, as that is its address
- Pointers are variables: thus they can be read, written and so on
- Being variables, they are stored in memory and they have an address as well

Pointers: Declaration and Initialization

- To declare a pointer, it is necessary to specify the type of the pointed object

```
int *thing_ptr;
```

- In this case `thing_ptr` will hold the address of an `int`
- The operator `&` returns the address of a variable

```
thing_ptr = &a;
```

Do We Need Pointers?

- Pointers are a critical aspect of C
- Extremely powerful, allow to solve quickly, efficiently and shortly difficult problems
- Extremely powerful: if you misuse them you can compromise the integrity of your program
  - it is the general source of crashes of programs
- Students usually do not like them, because the different meanings of $*$ is confusing
- You cannot claim you know C if you do not master them

Managing a Variable via a Pointer

- The following pieces of code do the same

```
1     int a = 10;
2     a = a + 5;
```

```
1     int a = 10;
2     int *ptr;
3     ptr = &a;
4     *ptr = *ptr + 5;
```

- Note the different meanings of `*`:
  - Declaration: `*` declares `ptr` as pointer to integer
  - Next line: `*` dereference (or content-) operator

The Dereference Operator (The Content Operator)

- The $*$ operator allows you to access the content of the pointed object (content-operator)
- In the previous example we do not increase the pointer, but the content of the pointed object: `a`
- Note that

```
1    int *ptr;
2    ptr = &a;
3    ptr = ptr + 5;
```

  is accepted by the compiler but has a completely different meaning!
  (the pointer itself will be increased by the size of 5 integers meaning $5 * 4 = 20$ bytes)

An Example

```
1   #include <stdio.h>
2   int main() {
3     int a = 7;
4     int *ptr;
5     ptr = &a;
6
7     printf("Address of a: %p\n", ptr);
8     printf("Value of a: %d\n", *ptr);
9     return 0;
10  }
```

Simple Use of Pointers

```
1  #include <stdio.h>
2  int main() {
3     int  thing_var; /* def. var. for a thing */
4     int *thing_ptr; /* def. pointer to a thing */
5     thing_var = 2; /* assign a value to a thing */
6     printf("Thing %d\n", thing_var);
7     thing_ptr = &thing_var; /* make the pointer
       point to thing_var */
8     *thing_ptr = 3; /* thing_ptr points to
       thing_var,so thing_var changes to 3 */
9     printf("Thing %d\n", thing_var);
10    printf("Thing %d\n", *thing_ptr);
11    /* another way */
12    return 0;
13 }
```

Comments

- It is obvious that it is necessary to specify the type of the pointed variable
  - Previous example: how many bytes for the integer: 2 of 4? Knowing the type the answer can be given
- The following does not work

```
1    int a = 45;
2    int *int_ptr = &a;
3    char *char_ptr;
4    char_ptr = int_ptr; /* WRONG */
```

Misuse of Pointers

- `*thing_var`
  - illegal: Get the object pointed to by the variable `thing_var`, `thing_var` is not a pointer, so operation is invalid
- `&thing_ptr`
  - legal, but strange. `thing_ptr` is a pointer: Get address of pointer to a object, result is a pointer to a pointer

Arrays (1)

- Arrays (or vectors) can be defined in different ways:
  - An indexed variable
  - A contiguous memory area holding elements of the same type
- Arrays are useful if you have to deal with many variables of the same type
  - Logically related to each other

The Need for Arrays

- Write a program that reads all the students grades and prints the maximum and minimum, the mean and the variance
  - Assume at most 200 students
- Should we declare 200 float variables?
- From a logical point of view, all the grades belong to the same set of data

Arrays (2)

- An array has a common name and a shared type
- An array has a dimension
- Elements are numbered sequentially
- Elements can be accessed (read/write) in an array by using their position (also called index or subscript)

Arrays in C

- In C you declare an array by specifying the dimension between square brackets

  ```
  int data[4];
  ```

- The former is an array of 4 elements
- The first one is at position 0, the last one is at position 3

```
1     data[0] = 5;
2     data[1] = 7;
3     data[2] = 3;
4     data[3] = 8;
```

Strings

- Strings are sequences of characters
- Many languages have strings as a built-in type, but C does not
- C has character arrays, with the special character '\0' to indicate the end of the string

```
1  #include <stdio.h>
2  int main() {
3    char name[30] = "Hello World";
4    printf("%s\n", name);
5    name[5] = '\0';
6    printf("%s\n", name);
7    return 0;
8  }
```

Reading Strings from the Keyboard

Standard function `fgets` may be used to read a string from the keyboard.

```
fgets(name, sizeof(name), stdin);
```

- `name`
  - the name of the character array
- `sizeof(name)`
  - maximum number of characters to read
- `stdin`
  - is the file/stream to read from
    `stdin` means standard input (keyboard)

Example Program

```
1   #include <string.h>
2   #include <stdio.h>
3   int main() {
4     char line[100];
5     printf("Enter a line: ");
6     fgets(line, sizeof(line), stdin);
7     printf("You entered: %s\n", line");
8     printf("The length of the line is: %s\n",
9       strlen(line));
10    return 0;
11  }
```

This example contains three errors, one syntax error, and two logical errors. One logical error might crash your program, the other will just count the number of characters you have put wrong.

Reading Numbers with `fgets` and `sscanf`

Use `fgets` to read a line of input and `sscanf` to convert the text into numbers

```
1   #include <stdio.h>
2   int main() {
3     char line[100];
4     int value;
5     printf("Enter a value: ");
6     fgets(line, sizeof(line), stdin);
7     sscanf(line, "%d", &value);
8     printf("You entered: %d\n", value);
9     return 0;
10  }
```

Reading Numbers with scanf

```
1  #include <stdio.h>
2  int main() {
3    int value;
4    printf("Enter a value: ");
5    scanf("%d", &value);
6    printf("You entered: %d\n", value);
7    return 0;
8  }
```

Problems with scanf (1)

```
1  #include <stdio.h>
2  int main() {
3     int nr;
4     char ch;
5     scanf("%d", &nr);
6     // getchar();
7     scanf("%c", &ch);
8     printf("nr=%d, ch=%c\n", nr, ch);
9     return 0;
10 }
```

Assuming that you want to input 12 for `nr` and `'a'` for `ch` then you will experience `nr=12` and `c='\n'` which is not correct, therefore using `scanf` is not advisable in this context

One solution to solve this is to remove the comments in line 6

Problems with `scanf` (2)

- "The function `scanf` works like `printf`, except that `scanf` reads numbers instead of writing them. `scanf` provides a simple and easy way of reading numbers that almost never works. The function `scanf` is notorious for its poor end-of-line handling, which makes `scanf` useless for all but an expert."
- "However we have found a simple way to get around the defiencies of `scanf`, we do not use it."

From: Steve Oualline, Practical C Programming, O'Reilly, 3rd edition