CH-230-A

Programming in C and C++

C/C++

Tutorial 3

Dr. Kinga Lipskoch

Fall 2021

Finding the Maximum Value in an Array

```
1 /*   v[100]: array of ints
2      dim: number of elements in v
3      Returns the greatest element in v
4 */
5 int findmax(int v[100], int dim) {
6   int i, max;
7
8   max = v[0];
9   for (i = 1; i < dim; i++) {
10     if (v[i] > max)
11       max = v[i];
12     }
13   return max;
14 }
```

Looking for an Element

```
1 /*   v[100]: array of ints
2      dim: number of elements in v
3      t: element to find
4      Returns -1 if t is not present in v or
5      its position in v
6 */
7 int find_element(int v[100], int dim, int t)  {
8   int i;
9   for (i = 0; i < dim; i++) {
10    if (v[i] == t)
11       return i;
12  }
13  return -1;
14 }
```

Flow of Execution

```
1 #include <stdio.h>
2
3 int main () {
4   int array [] = {2, 4, 8, 16, 32};
5   int result ;
6
7   result = find_element (array , 5, 37) ;
8   if ( result == -1)
9     printf ("37 is not present \n") ;
10
11   return 0;
12 }
```

Pointers and Address Arithmetic

- ▶ The arithmetic operators for sum and difference (+, -, ++, --, etc) can be applied also to pointers
  - ▶ After all a pointer stores an address, which is an integer
- ▶ These operators are subject to the "address arithmetic".
- ▶ Increasing a pointer means that the pointer will point to the following element
  - ▶ You can also add a number other than 1
- ▶ From a logic point of view the pointer is increased by one. From a physical point of view, the increment depends on the size of the pointed type

Address Arithmetic: Example (1)

```c
1  int main () {
2    char a_string[] = "This is a string\0";
3    char *p;
4    int count = 0;
5    printf("The string: %s\n", a_string);
6    for (p = &a_string[0]; *p != '\0'; p++)
7      count++;
8    printf("The string has %d chars.\n", count);
9    p--;
10   printf("Printing the reverse string: ");
11   while (count > 0) {
12     printf("%c", *p);
13     p--;
14     count--;
15   }
16   printf("\n");
17   return 0;
18 }
```

Address Arithmetic: Example (2)

```
1  int main () {
2    char a_string [] = "This is a string\0";
3    char *p;
4    int count = 0;
5    printf("The string: %s\n", a_string);
6    p = a_string;This would directly lead to 0% for the exercise!
7    while (*p != '\0') {
8      p++;
9      count ++;
10   }
11   printf("The string has %d characters.\n", count);
12   printf("Printing the reverse string: ");
13   p--;
14   while (count > 0) {
15     printf("%c", *p);
16     p--;
17     count --;
18   }
19   printf("\n");
20   return 0;
21 }
```

Increasing a Pointer will Increase the Memory Address Depending on the Size of Type

```c
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 int main() {
 4   char ch_arr[2] = {'A', 'B'};
 5   char *ch_ptr;
 6   float f_arr[2] = {1.1, 2.2};
 7   float *f_ptr;
 8
 9   ch_ptr = &ch_arr[0];        /* same as ch_ptr = ch_arr */
10   printf("%p\n", ch_ptr);   /* address of 1st elem */
11   ch_ptr++;                   /* increase pointer    */
12   printf("%p\n", ch_ptr);   /* address of 2nd elem */
13   printf("%c\n", *ch_ptr);  /* content of 2nd elem */
14
15   f_ptr = f_arr;              /* same as &f_arr[0]   */
16   printf("%p\n", f_ptr);    /* address of 1st elem */
17
18   f_ptr++;                    /* increase pointer    */
19   printf("%p\n", f_ptr);    /* address of 2nd elem */
20   printf("%f\n", *f_ptr);   /* content of 2nd elem */
21   return 0;
22 }
```

Predefined and User Defined Functions

▶ Predefined functions are functions provided by the language or by the host

▶ Operating system
  ▶ Library functions: they usually provide general purpose functionalities

▶ User defined functions are defined by the program
  ▶ Usually targeted to the problem being solved

Functions: Motivation

▶ Writing a 50000 lines long main function can be really difficult

▶ Splitting the code into many small pieces has many advantages:

  ▶ Easier to develop
  ▶ Easier to maintain and debug
  ▶ Increased opportunities to reuse the code

▶ An example: the printf function

  ▶ Developed by specialists
  ▶ Up to now we used it without knowing how it works internally
  ▶ Should there be a bug in it, by just using an updated version you can fix your code at once

Some Analogies

▶ A function can be thought as a mathematical function
▶ A function can be thought as a black box performing some
   functionality

Parameter 1 $\longrightarrow$

Parameter 2 $\longrightarrow$ function $\longrightarrow$ Return value

Parameter 3 $\longrightarrow$

0 to N parameters   **max. 1 return value**

Functions in C

- ▶ **Function declaration** (prototyping)
- ▶ **Function call** (use)
- ▶ **Function definition**
- ▶ Call should be preceded by prototyping (ANSI C (<u>A</u>merican <u>N</u>ational <u>S</u>tandards <u>I</u>nstitute) strongly advises this)
- ▶ There can be many declarations and many calls
- ▶ There must be exactly one definition

Prototyping

▶ The prototype is a statement declaring
   `return_type functionname(parameters);`
▶ Returned type is the type of the data
   ▶ may be empty, default type is `int`
   ▶ always declare the `return_type` explicitly
▶ Name follows the usual rules
▶ Parameters specify the number and types of the possible
   parameters
   ▶ may be empty
   ▶ always use explicit `void`, if function does not take arguments

The void Keyword

- void can be used to specify that
    - The function does not return any value
    - The function does not take any parameter
- int unknown(void);
    - function does not take any parameters
- int unknown();
    - function takes arbitrary number of parameters (to be compliant with the old Kernighan & Ritchie style)

Remember the Difference

- `void`
    - No return value
    - No parameter
- `void *`
    - Generic pointer (a pointer with no specific type which can be casted to any type)

Prototyping: Why?

▶ By having a prototype the compiler can check if the calls are performed correctly
   ▶ Number of parameters, types, etc.
▶ It is now clear why prototypes should always appear before calls

Prototypes: Examples

- ▶ Prototypes of functions in `math.h`
  ```
  double sqrt(double x);
  double pow(double x, double y);
  ```
- ▶ User defined function prototypes
  ```
  int find_max(int v[], int dim);
  void print_menu(char *options[], int dim);
  void do_something(void);
  ```
- ▶ `void` specifies no return value and empty parameters list

Function Definition

▶ The function definition specifies what a functions does

▶ Function definitions can contain everything (variables definitions, cycles, branches, etc) but NOT other function definitions

▶ A function terminates when
  ▶ it executes the last instruction
  ▶ it encounters a return statement

▶ Definition starts with the function header
  `return` type, name, parameters info

▶ Braces to define where the function starts and ends

▶ Business statements (instructions for carrying out the function's task)

What Happens when a Function is Called?

▶ The given parameters are copied into the corresponding entry in the parameters list

▶ The control is transferred to the function

▶ When the called function terminates, the control goes back to the caller function

Comment your Functions

▶ Every function should be commented
  ▶ Describe what the function does
  ▶ Describe each parameter (type and meaning)
  ▶ Describe what the function returns

▶ Look at the UNIX man pages to have an idea of how function
  documentation should look like
  man strcmp