Introduction to Programming in C

(IS-FEE-10061S)

Białystok University of Technology
Faculty of Electrical Engineering

Academic year 2023/2024

Workshop no. 12 (23.05.2024)

Jarosław Forenc, PhD

Topics

- Functions in C
  - general structure, arguments and function parameters
  - default function parameter values
  - function prototypes
  - passing vectors to functions

Program in C language

- A program in C language consists of functions and variables
  - functions contain statements that perform operations
  - variables hold values

```
#include <stdio.h>     /* diagonal of a square */
#include <math.h>

int main(void)
{
    float a = 10.0f, d;

    d = a * sqrt(2.0f);
    printf("Side = %g, diagonal = %g\n",a,d);

    return 0;
}
```

```
Side = 10, diagonal = 14.1421
```

Program in C language

- A program in C language consists of functions and variables
  - functions contain statements that perform operations
  - variables hold values

```
#include <stdio.h>     /* diagonal of a square */
#include <math.h>

int main(void)
{
    float a = 10.0f, d;

    d = a * sqrt(2.0f);
    printf("Side = %g, diagonal = %g\n",a,d);

    return 0;
}
```

function definition

Program in C language

- A program in C language consists of functions and variables
  - functions contain statements that perform operations
  - variables hold values

```c
#include <stdio.h>    /* diagonal of a square */
#include <math.h>

int main(void)
{
    float a = 10.0f, d;

    d = a * sqrt(2.0f);
    printf("Side = %g, diagonal = %g\n",a,d);

    return 0;
}
```

function calls

Functions

```c
#include <stdio.h>     /* diagonal of a square */
#include <math.h>
```

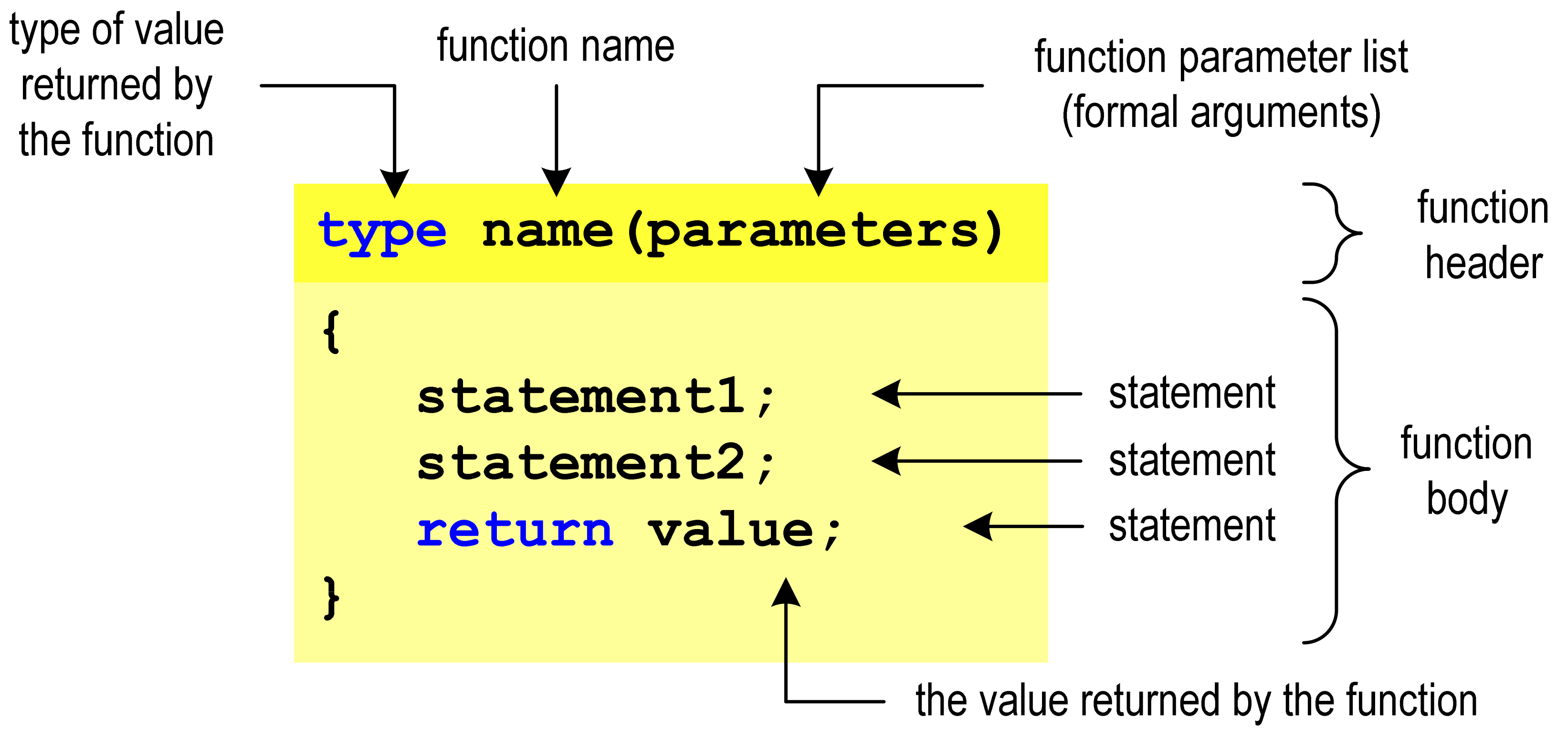
function definition

```c
float diagonal(float side)
{
    float result;
    result = side * sqrt(2.0f);
    return result;
}
```

function definition

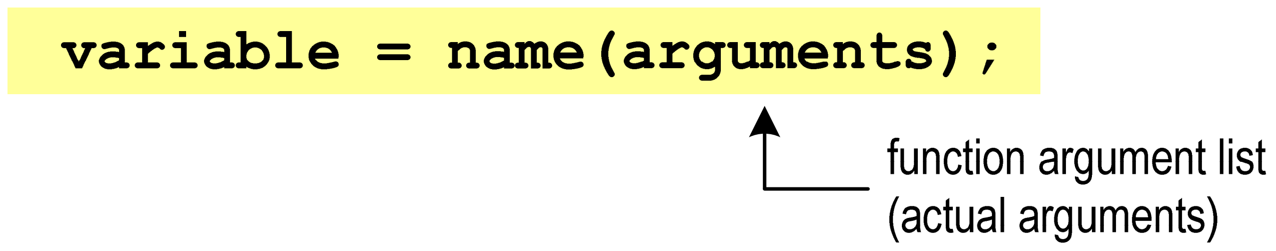
```c
int main(void)
{
    float a = 10.0f, d;
    d = diagonal(a);
    printf("Side = %g, diagonal = %g\n",a,d);
    return 0;
}
```

The general structure of functions in C



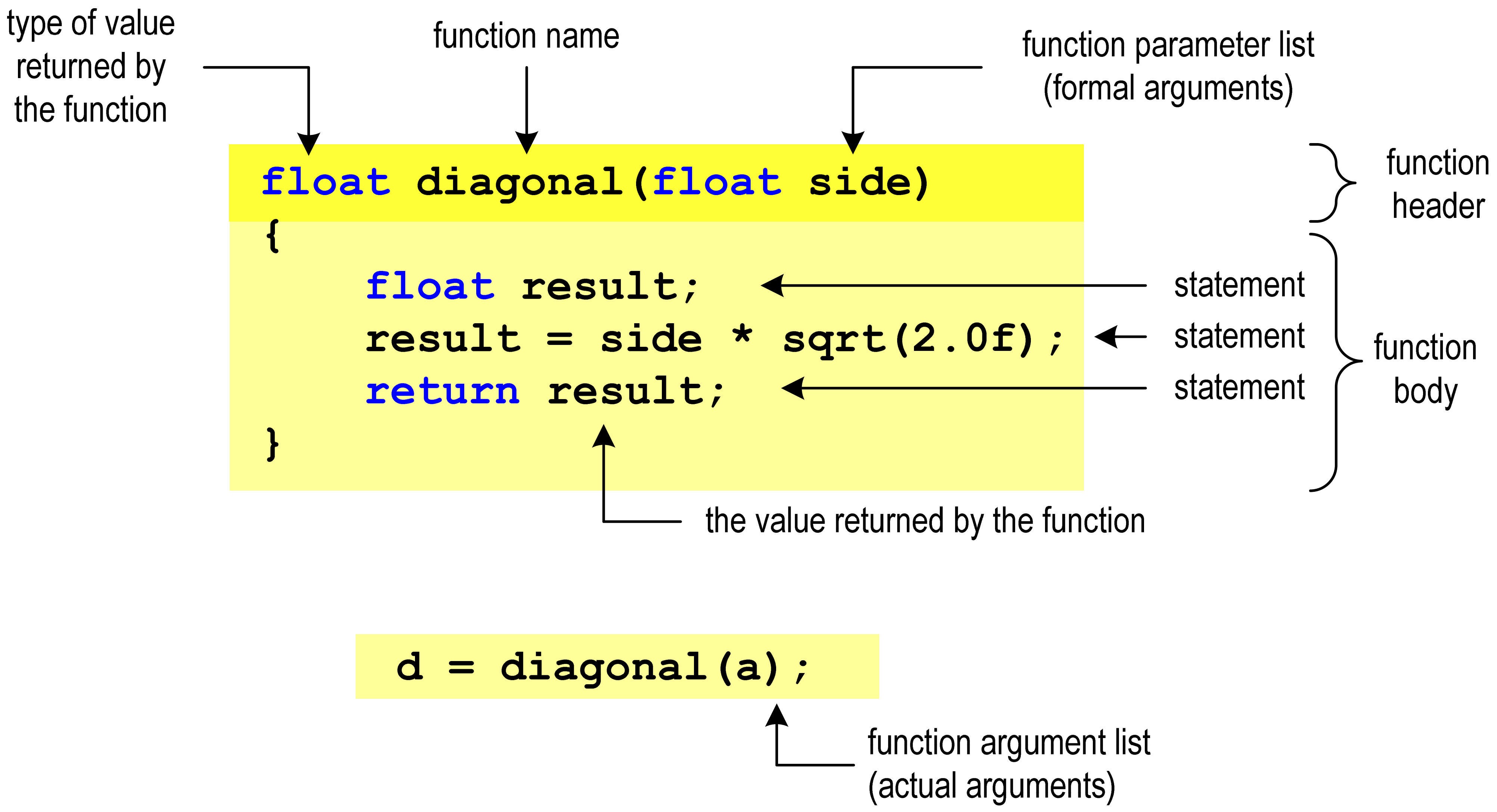
```
type name(parameters)
{
    statement1;
    statement2;
    return value;
}
```

type of value returned by the function
function name
function parameter list (formal arguments)
function header
statement
statement
statement
function body
the value returned by the function



```
variable = name(arguments);
```

The general structure of functions in C



```c
float diagonal(float side)
{
    float result;
    result = side * sqrt(2.0f);
    return result;
}
```

```c
d = diagonal(a);
```

type of value returned by the function
function name
function parameter list (formal arguments)
function header
statement
statement
statement
function body
the value returned by the function
function argument list (actual arguments)

Functions: arguments

- Function arguments can be numeric constants, variables, arithmetic expressions, or function calls

```
d = diagonal(a);
d = diagonal(10);
d = diagonal(2*a+5);
d = diagonal(sqrt(a)+15);
```

- A function call can be an argument to another function

```
printf("Side = %g, diagonal = %g\n",
                                a, diagonal(a));
```

Functions: parameters

- Parameters are treated in the same way as variables declared in this function and initialized with the values of the calling arguments

```
float diagonal(float side)
{
    float result;
    result = side * sqrt(2.0f);
    return result;
}
```

- The diagonal() function can be written in a simpler form:

```
float diagonal(float side)
{
    return side * sqrt(2.0f);
}
```

Functions: parameters

- If the function has several parameters, then for each of them the following is given:
    - parameter type
    - parameter name
- Parameters are separated by commas

```
/* diagonal of a rectangle */

float diagonal(float a, float b)
{
    return sqrt(a*a+b*b);
}
```

Functions: parameters

- Variables can have the same name in different functions

```
#include <stdio.h>     /* diagonal of a rectangle */
#include <math.h>

float diagonal(float a, float b)
{
    return sqrt(a*a+b*b);
}

int main(void)
{
    float a = 10.0f, b = 5.5f, d;

    d = diagonal(a,b);
    printf("Diagonal of a rectangle = %g\n",d);

    return 0;
}
```

Functions: default function parameter values

- Function parameters can have default values in function definition

```
float diagonal(float a = 10, float b = 5.5f)
{
    return sqrt(a*a+b*b);
}
```

- In this case, the function can be called with two, one, or no arguments

```
d = diagonal(a,b);
```

```
d = diagonal(a);
```

```
d = diagonal();
```

- Missing arguments will be replaced with default values

Functions: default function parameter values

- Not all parameters need to have default values
- Default values must be specified starting from the right side of the parameter list

```
float diagonal(float a, float b = 5.5f)
{
    return sqrt(a*a+b*b);
}
```

- The above function can be called with one or two arguments

```
d = diagonal(a,b);
```

```
d = diagonal(a);
```

- Default parameter values can be given in the declaration or in the function definition

Functions: the value returned by the function

- The return keyword can be used multiple times within a function

```
float grade(int pts)
{
    if (pts>90)                 return 5.0f;
    if (pts>80 && pts<91)       return 4.5f;
    if (pts>70 && pts<81)       return 4.0f;
    if (pts>60 && pts<71)       return 3.5f;
    if (pts>50 && pts<61)       return 3.0f;
    if (pts<51)                 return 2.0f;
}
```

91-100 pts. → 5.0	81-90 pts. → 4.5
71-80 pts. → 4.0	61-70 pts. → 3.5
51-60 pts. → 3.0	0-50 pts. → 2.0

Functions: prototype

- Is it possible to change the order of function definitions?

```
#include <stdio.h>     /* diagonal of a rectangle */
#include <math.h>

float diagonal(float a, float b)
{
    return sqrt(a*a+b*b);
}

int main(void)
{
    float a = 10.0f, b = 5.5f, d;
    d = diagonal(a,b);
    printf("Diagonal of a rectangle = %g\n",d);
    return 0;
}
```

function definition

function definition

Functions: prototype

- Is it possible to change the order of function definitions?

```
#include <stdio.h>    /* diagonal of a rectangle */
#include <math.h>

int main(void)
{
    float a = 10.0f, b = 5.5f, d;

    d = diagonal(a,b);
    printf("Diagonal of a rectangle = %g\n",d);

    return 0;
}

float diagonal(float a, float b)
{
    return sqrt(a*a+b*b);
}
```

function definition

function definition

Functions: prototype

- Is it possible to change the order of function definitions?

```c
#include <stdio.h>     /* diagonal of a rectangle */
#include <math.h>

int main(void)
{
    float a = 10.0f, b = 5.5f, d;
    d = diagonal(a,b);
    printf("Diagonal of a rectangle = %g\n",d);
    return 0;
}

float diagonal(float a, float b)
{
    return sqrt(a*a+b*b);
}
```

function definition

error C3861: 'diagonal': identifier not found

Functions: prototype

```
#include <stdio.h>     /* diagonal of a rectangle */
#include <math.h>
```

function prototype

```
float diagonal(float a, float b);
```

function definition

```
int main(void)
{
    float a = 10.0f, b = 5.5f, d;
    d = diagonal(a,b);
    printf("Diagonal of a rectangle = %g\n",d);
    return 0;
}
```

function definition

```
float diagonal(float a, float b)
{
    return sqrt(a*a+b*b);
}
```

Functions: prototype

- The prototype of a function is its header ending with a semicolon

```
float diagonal(float a, float b);
```

- Another name of the function prototype:
    - function declaration
- Through the prototype, the compiler verifies the following during a function call:
    - the function name
    - the number and types of arguments
    - the return type

```
d = diagonal(a,b);
```

- Parameter names are irrelevant and may be omitted:

```
float diagonal(float, float);
```

Functions: prototype

- Placing a function prototype and omitting its definition will result in an error not during the compilation stage, but during the linking stage

```
#include <stdio.h>    /* diagonal of a rectangle */
#include <math.h>

float diagonal(float a, float b);

int main(void)
{
    float a = 10.0f, b = 5.5f, d;
    d = diagonal(a,b);
    printf("Diagonal of a rectangle = %g\n",d);
    return 0;
}
```

function prototype

function definition

Functions: prototype

- Placing a function prototype and omitting its definition will result in an error not during the compilation stage, but during the linking stage

```
1>Compiling...
1>test.cpp
1>Compiling manifest to resources...
1>Microsoft (R) Windows (R) Resource Compiler Version 6.0.5724.0
1>Copyright (C) Microsoft Corporation.  All rights reserved.
1>Linking...
1>test.obj : error LNK2019: unresolved external symbol "float __cdecl
diagonal(float,float)" (?diagonal@@YAMMM@Z) referenced in function _main
1>D:\test\Debug\test.exe : fatal error LNK1120: 1 unresolved externals
```

Functions

- The functions presented so far had arguments and returned values
- The structure and calling of such a function are as follows

```
type name(parameters)
{
    statements;
    return value;
}
```

```
type var;
var = name(arguments);
```

- We can also define functions that take no arguments and/or return no value

Functions

- A function with no arguments and no return value:
  - in the function header, the return type is void
  - instead of parameters, the keyword void is used, or nothing is entered
  - if there is a return statement, it must not be followed by any value
  - if the return return statement is not present, the function terminates after all statements have been executed
- Function structure:

```
void name(void)
{
    statements;
    return;
}
```

```
void name()
{
    statements;
    return;
}
```

Functions

- A function with no arguments and no return value:
  - in the function header, the return type is void
  - instead of parameters, the keyword void is used, or nothing is entered
  - if there is a return statement, it must not be followed by any value
  - if the return return statement is not present, the function terminates after all statements have been executed
- Function structure:

```
void name(void)
{
    statemets;
}
```

```
void name()
{
    statements;
}
```

- Function call:

```
name();
```

Functions - example

```c
#include <stdio.h>

void print_line(void)
{
    printf("--------------------------------\n");
}

int main(void)
{
    print_line();
    printf("The functions are not difficult!\n");
    print_line();

    return 0;
}
```

```
--------------------------------
The functions are not difficult!
--------------------------------
```

Functions: arguments (vectors)

- When an array is passed to a function, no copy of it is created, and all operations on its elements refer to the array from the calling function
- The function header contains the type of array elements, its name, and square brackets with the number of array elements or just square brackets

```
void fun(int tab[5])
{
    ...
}
```

```
void fun(int tab[])
{
    ...
}
```

- When calling a function, only its name is provided, without square brackets

```
fun(tab);
```

Functions: arguments (vectors) - example

```c
#include <stdio.h>

void dispaly(int tab[])
{
    for (int i=0; i<5; i++)
        printf("%3d",tab[i]);
    printf("\n");
}

void zero(int tab[5])
{
    for (int i=0; i<5; i++)
        tab[i] = 0;
}
```

```c
float average(int tab[])
{
    float av = 0;
    int sum = 0;

    for (int i=0; i<5; i++)
        sum = sum + tab[i];

    av = (float)sum / 5;

    return av;
}
```

Functions: arguments (vectors) - example

```
int main(void)
{
    int tab[5] = {1,2,3,4,5};
    float av;

    dispaly(tab);

    av = average(tab);
    printf("Average of elements: %g\n", av);
    printf("Average of elements: %g\n", average(tab));

    zero(tab);
    dispaly(tab);

    return 0;
}
```

```
 1  2  3  4  5
Average of elements: 3
Average of elements: 3
 0  0  0  0  0
```

End of workshop no. 12

Thank you for your attention!