Fachschaftsrat Informatik

C-Lessons

Variables

Lecturers: Mirko Jantschke, Pascal Scholz

21. November 2018

Overview

1. Using functions
2. More on scopes
3. Recursion
4. Related Task

Using functions

Remember the main *function*?

```c
int main(void) {
    /* code happens */
    return 0;
}
```

Defining functions

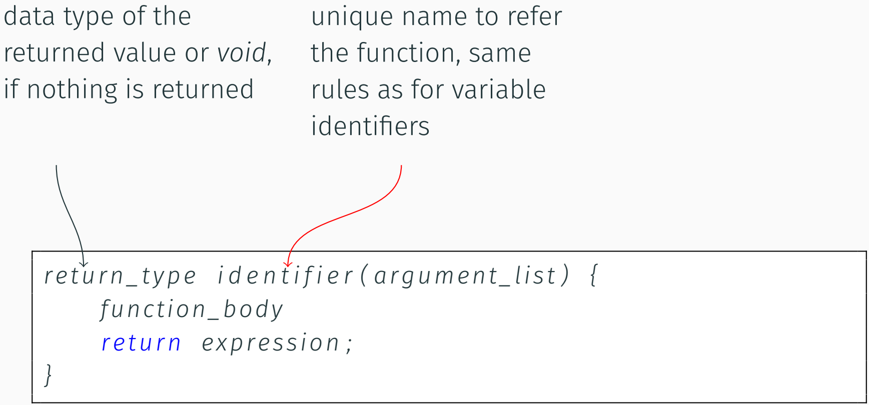
```
return_type identifier(argument_list) {
    function_body
    return expression;
}
```

Defining functions

data type of the
returned value or *void*,
if nothing is returned

```
return_type identifier(argument_list) {
    function_body
    return expression;
}
```

Defining functions

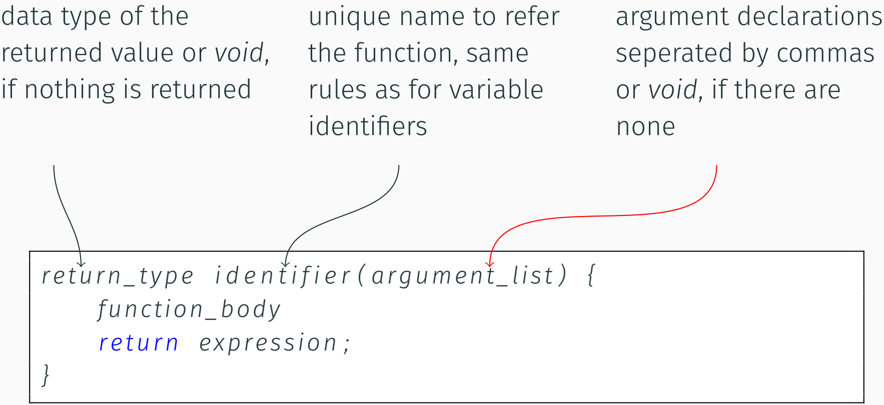


data type of the returned value or *void*, if nothing is returned

unique name to refer the function, same rules as for variable identifiers

```
return_type identifier(argument_list) {
    function_body
    return expression;
}
```

Defining functions



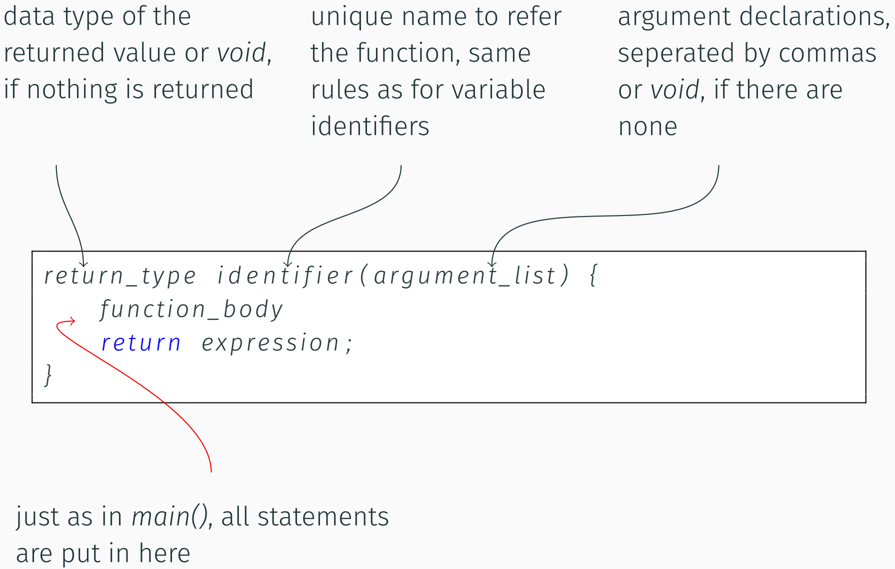
data type of the returned value or *void*, if nothing is returned

unique name to refer the function, same rules as for variable identifiers

argument declarations, seperated by commas or *void*, if there are none

```
return_type identifier(argument_list) {
    function_body
    return expression;
}
```

Defining functions



data type of the returned value or *void*, if nothing is returned

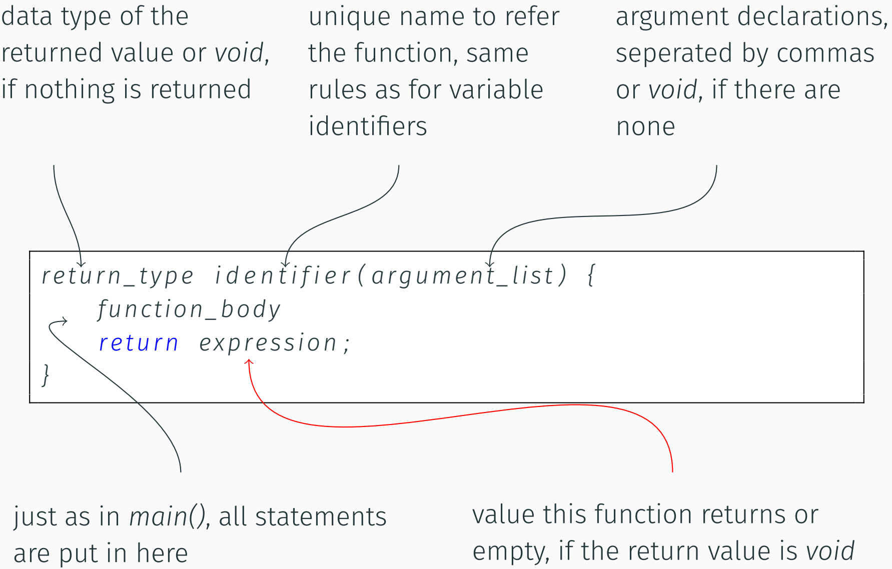
unique name to refer the function, same rules as for variable identifiers

argument declarations, seperated by commas or *void*, if there are none

```
return_type identifier(argument_list) {
    function_body
    return expression;
}
```

just as in *main()*, all statements are put in here

Defining functions



data type of the returned value or *void*, if nothing is returned

unique name to refer the function, same rules as for variable identifiers

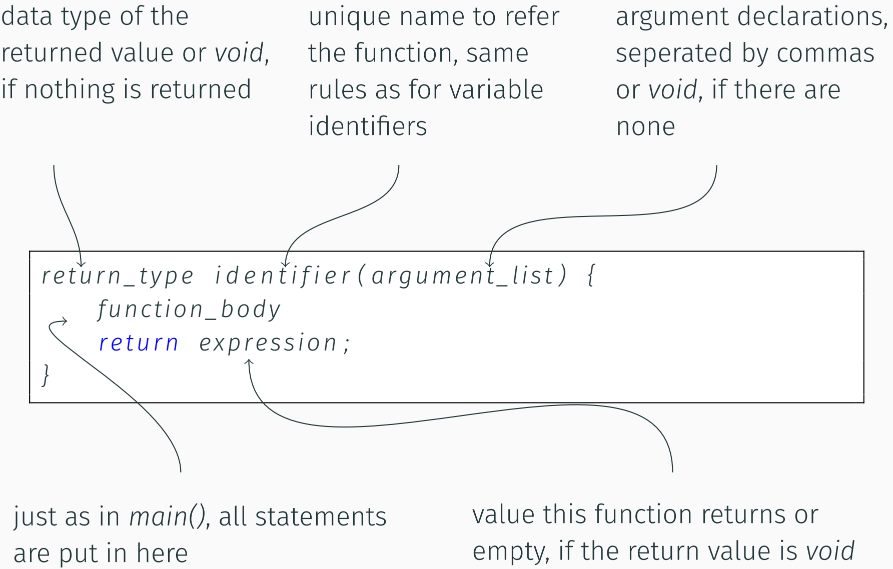
argument declarations, seperated by commas or *void*, if there are none

```
return_type identifier(argument_list) {
    function_body
    return expression;
}
```

just as in *main()*, all statements are put in here

value this function returns or empty, if the return value is *void*

Defining functions



data type of the returned value or *void*, if nothing is returned

unique name to refer the function, same rules as for variable identifiers

argument declarations, seperated by commas or *void*, if there are none

```
return_type identifier(argument_list) {
    function_body
    return expression;
}
```

just as in *main()*, all statements are put in here

value this function returns or empty, if the return value is *void*

Passing arguments

- Each value is assigned to the parameter at the same position in the argument list (and therefore must have the same type)

```c
#include <stdio.h>

void shift_character(char character, unsigned offset) {
    printf(”%c\n”, (character + offset) % 255);
}

int random_number(void) {
    return 4;   // chosen by fair dice roll.
                // guaranteed to be random.
}

int main(void) {
    int offset = 10;
    shift_character(’c’, offset);
    printf(”%d\n”, random_number());
    return 0;
}
```

More on scopes

Global variables

- Variables defined outside any function
- Scope: from line of declaration to end of program

```c
int globe = 42;

void foo(void) {
    globe = 23;
}

int main(void) {
    printf("%d\n", globe);  /* Prints 42 */
    foo();
    printf("%d\n", globe);  /* Prints 23 */
    ...
```

Altering them in one function may have **side effects** on other functions → use them rarely.

Where not to call functions

Since a function’s scope starts at the line of its definition, having two functions *f()* and *g()* calling each other is not possible:

```
void f(int i) {
    ...
    g(42);  /* What is g? */
}

void g(int i) {
    ...
    f(42);
}
```

In that case, *g()* is called outside its scope. Changing the order does not work either.

Prototypes

Like variables, functions can also be *declared*:

```
return_type identifier(argument list);
```

- It's similar to a definition, just replace the function body by a ;
- Declared functions must also be defined any where in the program
- In the argument list, only types matter → identifiers **can** be left out

```
void g(int i);  /* better do not leave the identifier out */

void f(int i) {
    ...
    g(42);      /* Now a call of g() can be compiled */
}

void g(int i) {...} /* g() definition, similar to f() */
```

Better program structure

To avoid problems like that above, it is a common practise to *declare* all functions at the top of the file and define them below the main function:

```c
void f(int i);
void g(int i);

int main(void) {
    ...
}

void f(int i) {
    ...
    g(42);
}

/* g() definition, similar to f() */
```

Good documentation style

Add a documentation comment to each function prototype:

```c
/*
 * Get the sum of two numbers.
 * num:    input number
 */
int factorial(int num);
```

There are frameworks such as *doxygen* that parse your comments and create a fancy HTML documentation:

```c
/**
 * @brief Get the sum of two numbers.
 * @param num1  first number
 * @param num2  second number
 * @return      sum of num1 and num2
 */
int add(int num1, int num2);
```

Functions in functions

You **could** define functions in functions.[1]

[1] Just saying.

Recursion

Recursive functions

- Functions calling themselves
- Used to implement many mathematical algorithms
- Easy to think up, but they run slow

Careful:

```c
void foo(void) {
    foo();
}
```

creates an infinite loop.[2]
There must always be an *exit condition* if using recursion!

[2] And, at some point, a program crash (*stack overflow*)

Exponentiation

As an example, take a look at this function calculating $base^{exponent}$:

```c
int power(int base, int exponent) {
    if (exponent == 0)
        return 1;
    return base * power(base, exponent - 1);
}
```

- $a^0 = 1 \rightarrow$ *power(a, 0)* just returns *1*
- $a^b = a \cdot a^{b-1} \rightarrow$ recursive call of *power(a, b-1)*

Example power(2,3)

```
int power(int base, int exponent) {
    if (exponent == 0)
        return 1;
    return base * power(base, exponent - 1);
}
```

1st call: power(2,3)

Example power(2,3)

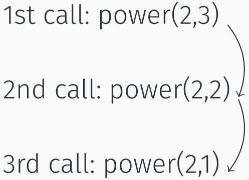
```c
int power(int base, int exponent) {
    if (exponent == 0)
        return 1;
    return base * power(base, exponent - 1);
}
```

1st call: power(2,3)

2nd call: power(2,2)

Example power(2,3)

```
int power(int base, int exponent) {
    if (exponent == 0)
        return 1;
    return base * power(base, exponent - 1);
}
```



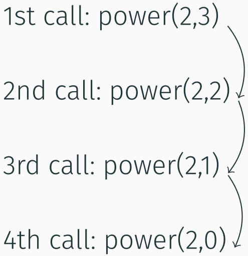
1st call: power(2,3)

2nd call: power(2,2)

3rd call: power(2,1)

Example power(2,3)

```c
int power(int base, int exponent) {
    if (exponent == 0)
        return 1;
    return base * power(base, exponent - 1);
}
```



1st call: power(2,3)

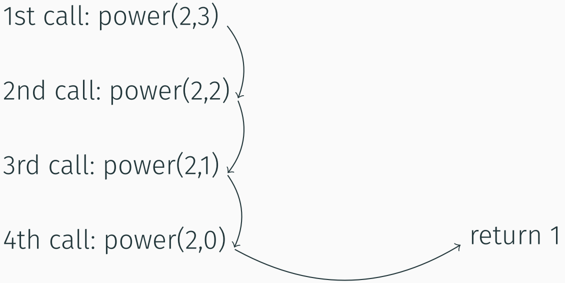
2nd call: power(2,2)

3rd call: power(2,1)

4th call: power(2,0)

Example power(2,3)

```
int power(int base, int exponent) {
    if (exponent == 0)
        return 1;
    return base * power(base, exponent - 1);
}
```



1st call: power(2,3)

2nd call: power(2,2)

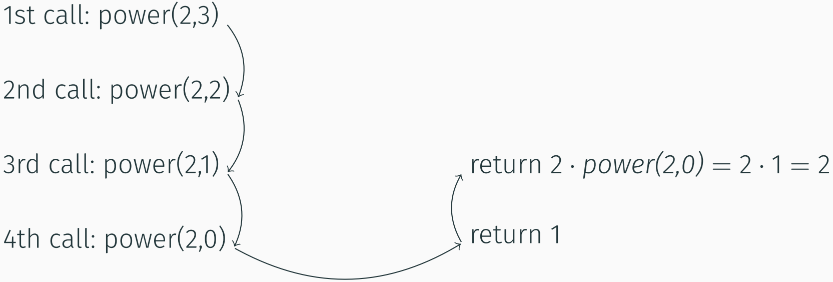
3rd call: power(2,1)

4th call: power(2,0)

return 1

Example power(2,3)

```c
int power(int base, int exponent) {
    if (exponent == 0)
        return 1;
    return base * power(base, exponent - 1);
}
```



1st call: power(2,3)

2nd call: power(2,2)

3rd call: power(2,1)

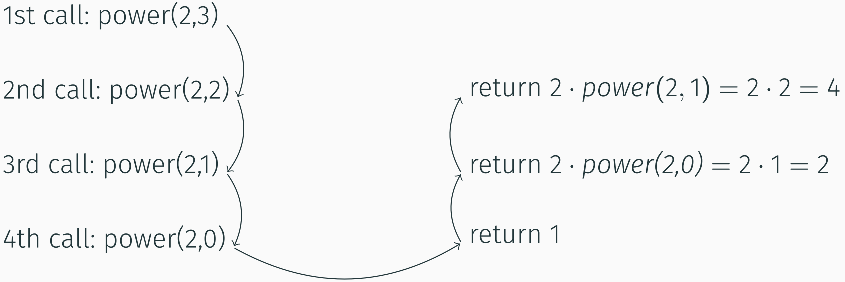
4th call: power(2,0)

return 2 · *power(2,0)* = 2 · 1 = 2

return 1

Example power(2,3)

```
int power(int base, int exponent) {
    if (exponent == 0)
        return 1;
    return base * power(base, exponent - 1);
}
```



1st call: power(2,3)

2nd call: power(2,2)

3rd call: power(2,1)

4th call: power(2,0)

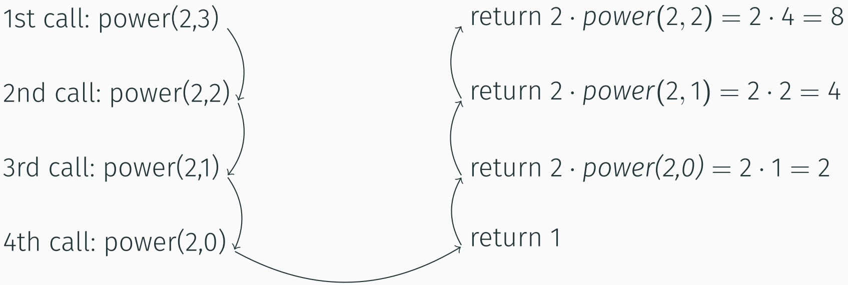
return $2 \cdot power(2,1) = 2 \cdot 2 = 4$

return $2 \cdot$ *power(2,0)* $= 2 \cdot 1 = 2$

return 1

Example power(2,3)

```
int power(int base, int exponent) {
    if (exponent == 0)
        return 1;
    return base * power(base, exponent - 1);
}
```



1st call: power(2,3)

2nd call: power(2,2)

3rd call: power(2,1)

4th call: power(2,0)

return $2 \cdot \mathit{power}(2,2) = 2 \cdot 4 = 8$

return $2 \cdot \mathit{power}(2,1) = 2 \cdot 2 = 4$

return $2 \cdot \mathit{power(2,0)} = 2 \cdot 1 = 2$

return 1

Related Task

Power and faculty

Task as online:
Write a function that takes a numbers a from the user and calculates a!

Experts: Write a program that calculates the fibonacci number of a fib(a) of the user input a.