Course
ELEC-A7100
Course materials
Your points
Microsoft Teams
Code Vault

This course has already ended.

« 1 Preprocessor Course materials 3 Function pointers »
ELEC-A7100 / 10. Advanced Features / 2 Variable length argument lists

Tenth round tasks as a `ZIP file.`

Variable length argument lists¶

Usually the *number* and *type* of arguments in C functions are fixed. However, there are situations when the exact number of parameters or their type cannot be determined. The most common example for this situation is the *printf*, and other functions that use format specifiers to handle variables. The C language contains a special handling mechanism for these cases.

A function can be defined with variable length parameter list with the following notation:

```
int printf(const char *fmt, ... )
```

The above is the function declaration of the *printf* function. As we know by now, the first parameter is always a string. After that, there is a variable number of other parameters, for which the type or number is not specified in function defintion. The implementation of *printf* determines the number of parameters based on the format string and the format specifiers included in it.

The parameter list will be processed using the **va_list** data type, and using macros **va_start**, **va_arg** and **va_end**. These are defined in the **stdarg.h** header.

Below is an example of function that calculates average from varying number of floating point numbers.

```
1  #include <stdio.h>
2  #include <stdarg.h>
3
4  double average(int parameter, ... )
5  {
6      va_list args;
7      double sum = 0;
8      va_start(args, parameter);
9      for (int i = 0; i < parameter; i++) {
10         sum += va_arg(args, double);
11     }
12     va_end(args);
13     return sum / parameter;
14 }
15
16 int main(void)
17 {
18     printf("average: %f\n", average(4, 1.0, 10.0, 0.1, 0.2));
19     printf("another: %f\n", average(2, 0.1, 0.3));
20
21     return 0;
22 }
```

va_start initializes the handling of the parameter list, and tells the argument after which the variable length list begins. In a function with a variable parameter list there always needs to be at least one fixed parameter.

The function picks the arguments one at a time using the *va_arg* macro. The macro takes the *va_list* instance as the first parameter, and the expected data type as the second parameter. The application logic therefore needs to have some way to determine this. In our example this is easy, because we know that all numbers are *double* type. On the other hand, the *printf* function determines the type of the next argument based on the format specified types in the format string. After all arguments have been processed, the *va_args* state needs to be cleaned up using *va_end* macro.

We can see from the *main* function that now we can call the *average* function with different number of parameters, as long as the function knows the number of arguments.

Task 10.2: Printer¶

Objective: Learn variable length argument lists

Implement function `int myprint(const char *str, ...)` that prints a variable number of integers to standard output stream following the format indicated by a given format string. The function can take variable number of arguments: **the first argument is always a (constant) format string** (as in *printf*), but the number of other arguments depends on the format string. Our output function is a simpler version of *printf*: it can only print integers. *myprint* differs from traditional *printf* by using character `&` as the format specifier that should be substituted by the integer indicated at the respective position in the argument list. Because we only output integers, this simple format specifier will suffice.

For example, this is one valid way of calling the function: `myprint("Number one: &, number two: &\n", 120, 1345);`

The function should return an integer, that indicates how many format specifiers (and integer arguments) were included in the given format string.

If your implementation works correctly, the main function will output the following:

```
Hello!
Number: 5
Number one: 120, number two: 1345
Three numbers: 12 444 5555
I just printed 3 integers
```

Hint: As a reminder, *strchr* will return pointer to the next instance of given character from a string, and *fputc* will output one character at a time (also to the standard output stream). You may or may not want to use these functions.

Points 25 / 25 My submissions 1
Deadline Friday, 13 August 2021, 19:59
To be submitted alone

This course has been archived (Saturday, 31 December 2022, 20:00).

My print

Select your files for grading

myprint.c

Choose File No file chosen

Submit