C

01-Introduction

Patrick Stiller

October 19, 2014

Overview

Organisation

- Proceeding
- Resources

Your first program

- Creating a new C-file
- Hello World

Basics

- Some definitions
- Calculating

About this course

Requirements

- You know how to use a computer
- You want to learn a imperative programming language

Proceeding

- There will be 10+ lessons
- Each covers a topic and comes with exercise

Some resources

- You can ask your tutor
- Join the Auditorium group
  https://auditorium.inf.tu-dresden.de/

Creating a C-file

We work with **plain text** editors like Gedit or Windows-Editor.

1. Open the editor
2. Save your file as : *helloworld.c*
3. Write your *C* program in there.

Minimal structure

```c
// a minimal C program

#include <stdio.h>

int main (int argc, char *argv[]){
  return 0;
}
```

Hello World

This is a small program printing *Hello World* to your console

```c
// Hello World !

#include <stdio.h> // needed for printf

int main (int argc, char *argv[]){
  printf("Hello World\n");
  return 0;
}
```

To find out more about how *printf* works:

Console: *man 3 printf*

Compile and run your files

1. Open your console
2. Move with the http://ss64.com/nt/cd.htmlcd-command to your workspace
3. Build with:
   gcc -o helloWorld helloWorld.c
4. Run with:
   ./helloWorld

About the gcc options

gcc has a lot of options.

Recomended are:
- `-Wall` enables all compiler's warning messages
- `-Wextra` enable extra warning messages
- `-Werror` make all warnings into errors.
- `-std=c99` set the c programming language standard
- `-pedantic` reject all programs that use forbidden extensions

Comments

```c
// Hello World!
#include <stdio.h>

/* a little program printing
 * -------HELLO WORLD-----
 */
int main (int argc, char *argv[]){
  printf("Hello World \n");
  return 0;
}
```

You should allways comment your code.
Code is read more often than it is written.

- // single line comment
- /* comment spanning
  multiple lines */

Primitive data types

C supports some primitive data types:

- char an ASCII character
- int at least an 8 bit integer
- long at least a 32 bit integer
- long long at least a 64 bit integer
- float at least a 32 bit floating point number
- double at least a 64 bit floating point number

Caution!

The real size of your data types depends on your hardware.
Never make assumptions about that.

Using of sizeof

The sizeof statement calculates the size of any datatype.
Result : The size of the datatype in bytes

```c
#include <stdio.h>
int main (int argc, char *argv[]){
  char c;
  printf("Size of char %zu \n",sizeof c);
  printf("Size of int %zu \n", sizeof(int));
  return 0;
}
```

Primitive data types II

To set the size of the data types use the types provided by

```c
#include <stdint.h>
```

For example:

- int8_t
- int16_t
- int32_t
- int64_t

Blocks

```c
#include <stdio.h>
// prints "hello world" on your console
int main (int argc, char *argv[]){
  printf("Hello World \n");
  return 0;
}
```

Everything between { and } is a block.
Blocks may be nested.

About the semicolon

```c
#include <stdio.h>

int main (int argc, char *argv[]){
  printf("Hello World \n");
  return 0;
}
```

Semicolons conclude all statements.
Blocks do not need a semicolon.

Naming of Variables

- The name of variables can begin with any letter or underscore.
- Usually the name starts with a small letter.
- Compound names should use `camelCase`.
- Use meaningful names.

```
#include <stdio.h>

int main (int argc, char *argv[]){
  int i = 0; // not very meaningful
  float number = 5.3; // also not meaningful
  int count = 0; // quite a good name
  int numberOfRotations = 1; // there you go
return 0;
}
```

Calculating with int I

```c
#include <stdio.h>
int main (int argc, char *argv[]){
  int i; // declare variable i
  i = 42; // assign  42 to variable i
  printf("Value of i : %d ", i); // prints 42
  i = i + 2; // addition
  printf("Value of i:%d ",i); // prints 44
  return 0;
}
```

After the assignment the variable is intialized.
Do not forget to assign your variables to avoid errors.

Calculating with int II

```c
#include <stdio.h>
int main (int argc, char *argv[]){
  int a =-2; // declaration and assignment of a
  int b; // declaration of b
  printf("Value of a : %d \n", a); // prints -2
  b = a; //assignment of b
  printf("Value of b : %d \n", b); // prints -2
  a++; // increase a
  printf("Value of a : %d \n", a); // prints -1
  b--;//decrease b
  printf("Value of b : %d \n", a); // prints -3
  return 0;
}
```

Calculating with int III

```c
#include <stdio.h>
int main (int argc, char *argv[]){
  int a = -2; // declaration and assignment of a
  int b = 3;   // declaration and assignment of b
  a = a + b;   // addition
  a = a - b;   // subtraction
  a = a * b;   // multiplication
  a = a / b;   // division
  a = a % b;   // modulo
}
```

work with floats and doubles

To write floats and doubles with decimal points use ”.” as opposed to ”,”

```c
#include <stdio.h>
int main (int argc, char *argv[]){
  float f = 10.12; // declaration and assignmment of f
  double d = 41.5; // declaration and assignmment of d
}
```

Floats use simple precision and Doubles use double precesion. The type of *C* encoding uses a sign,a significand,and an exponent. With this encoding, you can never guarantee that you will not have a change in your value.

References

- *C*-Reference `http://en.cppreference.com/w/c`
- Wikibook `http://en.wikibooks.org/wiki/A_Little_C_Primer`
- Indent style `http://en.wikipedia.org/wiki/Indent_style`
- Galileo Book `http://openbook.galileocomputing.de/c_von_a_bis_z/index.htm`