C-Course

Offered to you by I.C.T.S.V. Inter-*Actief*

Noah Goldsmid

Wednesday 11 October 2017

Table of Contents

What is C?

- Programming Language
  - Imperative
  - Static
  - Weakly typed
- Originated in 1969
- Different standards
  - C11 is the most recent

Syntax

- Java's syntax is based on C

```
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

Comments

- Same as Java comments
- // – Single line comment
- /* This is a
  Multiline comment */

Statements

- Very similar to Java
- { … } //Compound statement
- if (condition) {

```
    // statement
} else {
    // statement
}
```

- while (condition) …

Expressions

- Again very similar to Java
- `1 + 1;`
- `True && ( false || true );`
- `0xf0 >> 2;`
- `atoi("100");`
- `(1 < 2) ? true : false`

Variables

- `int i = 3;`
- Types:
  - `char`
  - `short`
  - `int`
  - `long`
  - `long long`
  - `float`
  - `double`
  - `long double`
- Most types can be prefixed with a signed/unsigned
- No String type! We will come back to that..

Arrays

- Syntax is similar to Java
  - But no need to instantiate them!
- Java:
  - `int[] myint = new int[16];`
  - `myint[4] = 1;`
- C:
  - `int myint[16];`
  - `Myint[4] = 1;`
- Small difference: stack vs heap allocation
  - Will get back to that...

Literals

- `int a = 16;`
- `int b = 0x16; //hex`
- `int c = 0x10;`
- `int d = 016; //octal`
- `int e = 020;`
- `int f = 018;`
- `long g = 16l;`
- `int h = 16u; //unsigned`
- `float i = 16.0f; //float`
- `float j = 16.0;`
- `float k = 16E0;//16*10^0`
- `char m = 'm';`
- `char n = '\n';`
- `char s[] = "abc";`

Preprocessor

- Processes the source before the compiler

- `#define ERR_STR "Error: PEBKAC"`
  - `puts(ERR_STR);`
- `#include <stdio.h>`
- `#define SUM(a,b) a+b`
  - `SUM(1,2)*5;`

Man 3

Evaluate the difference the bash commands:

- `man printf`
- `man 3 printf`
  - `Section 3 is library calls`
- `See man man for all sections and more info`
- `Man 3 [function_name]  can be very useful`

Toolchain

- Preprocessor
  - Processes the source code and resolves all preprocessor directives
  - #include, #define, everything that starts with #
- Compiler
  - Parses the code and compiles it to machine code
  - Compiles every file individually
- Linker
  - Links the different compiled files together into one executable

Hello, world!

```
#include<stdio.h>
int main()
{
   puts("Hello, world!\n");
   return 0;
}
```

- Compile this code using:
  `gcc -std=c11 -Wall hello.c -o hello`
- `Run using:`
- `./hello`

Pointers

- `int a = 100; //an integer`
- `int *b; //a pointer to an integer`
- `int **c; // a pointer to a pointer to an integer`
- `b = &a; // &a = the pointer to a`
- `c = &b; // c now points to b which points to a`
- `*b = 200;  // a = 200`
- `**c = 300; // a = 300`

Arrays

- `int a[16]; //an integer array of length 16`
- An array is a fancy pointer to the first element
- A points to the first element in the array
  - `a == &a[0]`

- `a[0] = 100; //Like in Java`
- `*a = 200;  // a[0] = 200`
- `a[idx]` is equivalent to `*(a+idx)`

Strings

- A string is an array of characters
- Terminated by a NULL character `'\0'`
- `char *string1 = "bla";`
- `char string2[] = "bla";`
- `string1 = "blabla"; //allowed`
- `string2 = "blabla"; //not allowed`

Standard Library

- Most of the work has been done
- Lots of info in the man pages
- Good starting points:

```
man 3 stdio

man 3 string
```

- `When in doubt, google!`

Exercises

- Exercise 2: Implement and test the functions.
- Exercise 3: Make a simple calculator

Functions and Prototypes

- Use to declare a function before you define it.
- Used to write libraries
- `int add(int x, int y);`
- ```
  int add(int x, int y)
  {
      return x + y;
  }
  ```
  ```

Memory allocation

- Stack:
  - Memory that is limited to the current function scope
  - Everytime you enter a function the stack ‘grows’
  - When you leave the function the stack ‘shrinks’ again
  - Limited in size
- Heap:
  - Available in the whole program
  - Manual management
  - Useful for large chunks of memory
  - Survives when the function ends

Heap

- All examples so far used the stack
- Memory on the heap is manually managed
- Allocated by calling malloc(size); //returns a pointer to the new address
- This memory will keep existing until free(address) is called

Heap

```
char * getHelloString() {
    char *mystr = malloc(16);
    strcpy(mystr, “Hello”);
    return mystr;
}
int main() {
    char* hello = getHelloString();
    puts(hello);
    free(hello);
    return 0;
}
```

Pitfalls

- Buffer overflow
- Undefined behaviour
- Strings are “one bigger” than their size
  - Termination character
- Memory leaks
- Do not ignore the compiler warnings
- Segfaults
  - Accessing memory you do not have access to
  - Can easily happen when dealing with pointers

After the course

- Use google
- `Use cppreference, it has a great c section`
  - `http://en.cppreference.com/w/c`
- Read the man pages!
- Try `gdb` the GNU debugger
  - Steep learning curve, but very worth it