Yale-NUS College - 2018-2019

Bruno Bodin

YSC2227: INTRO TO C

week 01.4.addon (auto-generated)

COMPILERS

- Turn C language into machine language
- For example `gcc filename.c` or `clang filename.c`

PREPROCESSING MACROS

- with gcc `-E` generates the C file after preprocessing
- examples:
  - `#define <TO_REPLACE_THIS> <BY_THAT>`
  - `#include <LIBRARY_HEADER_FILE>`

COMPILATION FLAGS

- Find error before they may hapend: -Wall -Werror
- *Optimise* your program: -O3
- Help *debugger* : -g
- Compile a library: -c
- Specify the output filename: -o <filename>

COMMAND-LINE ARGUMENTS

- When you run a program (`./a.out`) you can provide arguments
- These arguments are accessible via the main function :

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

- `argc` the number of *arguments* , `argv` an `array of strings` .
- the return value of main is the return value of the program `$?`
- Anywhere in the program `exit(v)` can also terminate the program and return v.
- How can you determine the end of a string?

TERNARY OPERATOR

- Keyword: **?:**

```
<condition> ? <if true> : <if false>
```

- Example:

```
int a = test ? 0 : 1 ;
```

CASTING

it is possible to consider the value of a variable as if it was from a different type. This is **casting**. For example :

```
int   a  = (int) 'a' ;
float b  = (float) a ;
int   c  = (int) (b / 2);
float d  = (float) (a / 2);
float e  = (float) a / 2;
int   f  = (int  )  b;
int * fp = (int *) &b;

printf("%d␣%f␣%d␣%f␣%f␣%d␣%d\n",a, b, c, d, e, f, *fp);
```

SWITCH-CASE

- Keyword: **switch** and **case**

```
switch (<variable>) {
  case <value1> :
  // starts here if variable == value1
  case <value2> :
  // otherwise starts here if variable == value2
  default :
  // otherwise starts here
}
```

- if you jump to a `case` all the following `case` will be executed, you can use `break` at the end of `cases` to avoid that.

BREAK AND CONTINUE

- `break` within a `loop` or a `switch-case` interrupts it.
- `continue` within a `loop` interrupts the current iteration and go to the next one

CONST KEYWORD

- This keyword protect variable from writing, they can only be initialized.
- Example :

```
const int a = 0;
a++; // does not compile
```

- However this is not error-prone, this is just a hint to the compiler
- Example :

```
const int a = 0;
(*((int*)(&a)))++; // does compile, but may crash depending of O.S.
```

ARRAY

- Three ways to define an array:

```
<type> <name> [ <number of elements> ] ;
<type> <name> [ <number of elements> ] = { <value 0>, <value 1>, ...};
<type> <name> [                      ] = { <value 0>, <value 1>, ...};
```

- Then can access elements using brackets, like with a pointer :

```
char name [ 3 ] = {'c', 'a', 't'} ;
printf ("%c%c%c", name[0], name[1], name[2]);
```

- some syntax sugar with strings :

```
char name [ ] = "cat" ;
printf ("%c%c%c", name[0], name[1], name[2]);
```

- What is the value of `name[3]` ?

SCOPE

- **Global scope: entire program**

```
int a; // in a source file
```

then in a different source file

```
extern int a; // can retreive this variable
```

please note

```
static int a; // stop extern from doing so
```

- **Local scope: limited to current block**

```
{ int a = 0; }
{ int b = 0; b++; }
{ int b = 0; a++; } // raise an error
```

EXERCISE

- Create a program with two char arrays

```
char string1[] = "Why would you do that?";
char string2[] = "WhY WoUlD YoU Do tHaT?";
```

- With `sizeof` you can get the length of this string, but be careful this is not its purpose,
- For example what happen if you try `sizeof(str)` when `char str[]` is a function's argument ?
- Create a function `size_t strlen(const char *)`
- Create a function to test whether these two arrays are equal (case sensitive).
- Create a case insensitive version.

ASSIGNEMENT

In the ASCII standard, letters are numbers[1]. For example `printf("%x",'a');` would print 61.

Implement the following functions :

- `int islower (int c);` return `TRUE`[2] if and only if c is a lowercase letter (a to z).
- `int isupper (int c);` return `TRUE` if and only if c is an uppercase letter (A to Z).
- `int isalpha (int c);` return `TRUE` if and only if `isupper(c)` or `islower(c)` would return `TRUE`.
- `int isdigit (int c);` return `TRUE` if and only if c is a digit [0-9].
- `int isalnum (int c);` return `TRUE` if and only if `isalpha(c)` or `isdigit(c)` would return `TRUE`.
- `int iscntrl (int c);` return `TRUE` if and only if c is a control character.[3]

[1] https://simple.m.wikipedia.org/wiki/File:ASCII-Table-wide.svg

[2] `TRUE != 0`

[3] control characters are those between ASCII codes 0x00 (NUL) and 0x1f (US), plus 0x7f (DEL).

ASSIGNEMENT

In the ASCII standard, letters are numbers[1]. For example `printf("%x",'a');` would print 61.

Implement the following functions :

- `int islower (int c);` return `TRUE` if and only if c is a lowercase letter (a to z).
- `int isupper (int c);` return `TRUE` if and only if c is an uppercase letter (A to Z).
- `int isalpha (int c);` return `TRUE` if and only if c is a letter (a to z or A to Z).
- `int isdigit (int c);` return `TRUE` if and only if c is a digit (0 to 9).
- `int isalnum (int c);` return `TRUE` if and only if `isalpha(c)` or `isdigit(c)` would return `TRUE`.
- `int iscntrl (int c);` return `TRUE` if and only if c is a control character.[3]

Easy to find on Internet, but then I cannot judge your level ...

[1] https://simple.m.wikipedia.org/wiki/File:ASCII-Table-wide.svg

[2] `TRUE != 0`

[3] control characters are those between ASCII codes 0x00 (NUL) and 0x1f (US), plus 0x7f (DEL).

TOPICS COVERED

- Variables and Assignement Operators ✓ (T. Bailey, Chapter 1 and 2)
- Numeric Data Types and Conversion ✓ (T. Bailey, Chapter 2)
- Arrays ✓ (T. Bailey, Chapter 8)
- Arithhmetic and bitwise operators ✓ (T. Bailey, Chapter 2 and 12)
- Compilation, flags, and command-line arguments ✓ (D. Harris C.10)
- Pointers ✓ (T. Bailey, Chapter 7)
- C functions ✓ (T. Bailey, Chapter 4)
- Files and I/O
- Control structures, logic operators, and loops ✓ (T. Bailey, Chapter 3)
- Scope ✓ (T. Bailey, Chapter 5)
- Structures and Unions
- Memory management and segmentation
- Basic libraries
- Makefile
- Debugging

KEY POINTS

- *preprocessor, compiler, linker*
- *preprocessing macro*
- *flags, optimise, debugger*
- *arguments*

REFERENCES

- cook and magician:
  https://pixabay.com/en/users/graphicmama-team-2641041/