C

lucky777

Reminder

First program

```c
#include <stdio.h>
#include <stdlib.h>

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

- `#include`
  The files we need to run the software. (stdio is required to print text on the screen)

- `int argc, char *argv[]`
  (These arguments are optional)
  When we run the software in a terminal (`./test`) we can give it arguments.
  These arguments are stocked in the `argv` array, and the number of arguments is stocked in `argc`.
  (The first argument is the name of the file we have run)

> For example : `./test lucky 777`
> `argc` = **3**
> `argv[]` = {"**test**", "**lucky**", "**777**"}

- `printf();`
  The function to print text on the standard output (screen).
  (This function requires the library `<stdio.h>`)

- `return 0;`
  (this line is optional)
  In **C**, the main method will return **0** if no problem occurred during the execution.

Comma truncation

```c
double ko;
ko = 5/2;
//ko = 2

double ok;
ok = 5.0/2.0
//ok = 2.5
```

Global and static

```c
#include <stdio.h>

int w;
static int x;

static void test();

int main() {
    int y;
    static int z;
    return 0;
}

static void test() {
    printf("Lucky world!\n");
}
```

- `int w;`
  - `w` is accessible from anywhere.

- `static int x;`
  - `x` is accessible from the current file.

- `int y;`
  - `y` is only accessible inside the function.

- `static int z;`
  - `z` is only accessible inside the function, but will keep it's value.

- `static void test()`
  - `test()` is only callable by a function inside the current file.

Switch case

```
unsigned langNumber;
printf("How many languages do you know? : ");
scanf("%d", &langNumber);

switch (langNumber)
{
case 0:
  printf("Wait what?");
  break;
case 1:
  printf("I guess you speak english");
  break;
case 2:
  printf("You are bilingual");
  break;
case 3: //Will
case 4: //Execute
case 5: //This line:
  printf("Wow! Nice");
  break;
default:
  printf("Amazing!!");
  break;
}
```

Ternary condition

```
if (isOk) {
    result = "ok";
} else {
    result = "error";
}

// Same as:
result = (isOk) ? "ok" : "error";
```

Prototypes

```c
#include <stdio.h>

test();

int main() {
    test();
    return 0;
}

void test() {
    printf("ok");
}
```

- As the `main()` is before the function `test()`, the compilator will think that `test()` doesn't exist.
  So we can add the prototype `test();` at the beginning of the code, to tell gcc that we will use a function called `test()` after.
  (If we put the `main()` at the end of the code, prototypes are not required)

Headers

main.c

```c
#include "test.h"

int main() {
    testFunction();
}
```

test.c

```c
#include "test.h"

void testFunction() {
    printf("Lucky world!\n");
}
```

test.h

```c
#ifndef TEST

#define TEST
#include <stdio.h>
void testFunction();
```

```
#endif
```

- **main.c**
  We include `test.h`

- **test.h**
  If **TEST** is not defined yet:
  - We define it
  - We include `<stdio.h>`
  - We call the prototype of `testFunction()`.

(The `#ifndef` stops at `#endif`)

- **test.c**
  We include `test.h` and write the codes of `testFunction()`.
  (`<stdio.h>` has already been included in `test.h`)

Note:

- `< >` is for standard libraries in `/usr/include/` directory *(on linux)*
- `" "` is for custom headers inside the current directory

Pointers

```c
#include <stdlib.h>
#include <stdio.h>

void f(int *i) {
  printf( "%d\n", *i); //2
  (*i)++;                       //   *i = value at address i
  printf("%d\n", *i);  //3
}

void g(int i) {
  printf( "%d\n", i);  //3
  (i)++;                        //   *i = value at address i
  printf("%d\n", i);   //4
}

int main() {
  int i = 2;
  f(&i);                        //   &i = address of i
  g(i);
  printf("%d\n", i);    //3
}
```

- `int i = 2;`
  Declaration of a variable `i` with the value `2`.
  This variable will have a random address.

- `f(&i)`
  Calls the function `f()` with the argument `&i`.
  - `i` = value of variable i
  - `&i` = the address of the variable i

- `void f(int *i)`
  Function `f()` with parameter of type `pointer`.
  A pointer is a variable type that points at an address.
  - `i` = address
  - `*i` = value at the address `i`

We give the address of `i` as parameter when we call `f(&i)` (and not the value), so when we change the value contained at the address `i` in the function, it will change the value of `i` everywhere else.

But when we call `g(i)`, we give the value of `i` as parameter, so it won't change `i` itself outside the function.