Yale-NUS College - 2018-2019

Bruno Bodin

YSC2227: INTRO TO C

week 01.3.c (auto-generated)

WHAT IS A C PROGRAM?

- A C program is a **humanly-readable** text file, we also talk about *source file* .
- It follows a syntax which only accepts **variable**, **functions**, and **preprocessing** instructions.
- Each source file can be compiled to any compatible architecture (*portability*), and they can also be combined (*modularity*).

```
#include <stdio.h>

static const char *string =  "hello␣world\n" ;

int main( )  {
  printf(string);
  return 0;
}
```

TYPES AND VARIABLES

- Types indicate (to the compiler) how to treat a particular data.
- Two fundamental *types* defined in C are "int" and "char".

[1] int has a maximal value

TYPES AND VARIABLES

- Types indicate (to the compiler) how to treat a particular data.
- Two fundamental *types* defined in C are "int" and "char".

- *Variables* are defined by a **type** and an ***identifier***.
- A valid syntax is `<type> <identifier>;`
- For example: `int a;` ($a \in \mathbb{Z}$ but not entirely true[1])

[1] int has a maximal value

TYPES AND VARIABLES

- Types indicate (to the compiler) how to treat a particular data.
- Two fundamental *types* defined in C are "int" and "char".

- *Variables* are defined by a **type** and an ***identifier***.
- A valid syntax is `<type> <identifier>;`
- For example: `int a;` ($a \in \mathbb{Z}$ but not entirely true[1])

- Variables can be ***initialized*** with a *value*.
- The syntax is `<type> <identifier> = <value>;`
- For example: `int b = 2;` ($b = 2 | b \in \mathbb{Z}$)

types, variables, identifier, initialized, value

[1] int has a maximal value

FUNCTIONS

- Functions are defined by an **identifier** , a **return type**, **inputs variables**, and a **body**.
- The *function body* is a sequence of *expressions* that specify the computation to be done.

```
<return type> <identifier> (<type1> <id1>, <type2> <id2>, <typeN> <idN>){
  expression1;
  expression2;
}
```

function body, expressions, keyword, return value

FUNCTIONS

- Functions are defined by an **identifier** , a **return type**, **inputs variables**, and a **body**.
- The *function body* is a sequence of *expressions* that specify the computation to be done.

```
<return type> <identifier> (<type1> <id1>, <type2> <id2>, <typeN> <idN>){
  expression1;
  expression2;
}
```

- The "return" *keyword* defines the *return value* of a function

```
int add(int  l,int r) {
  return  l +  r;
}
```

function body, expressions, keyword, return value

FUNCTIONS

- Functions are defined by an **identifier** , a **return type**, **inputs variables**, and a **body**.
- The *function body* is a sequence of *expressions* that specify the computation to be done.

```
<return type> <identifier> (<type1> <id1>, <type2> <id2>, <typeN> <idN>){
  expression1;
  expression2;
}
```

- The "return" *keyword* defines the *return value* of a function

```
int add(int  l,int r) {
  return  l +  r;
}
```

- The starting point of a C program is always the main function.

```
int main  () { ... }
```

function body, *expressions*, *keyword*, *return value*

LIVE DEMO

How to program a program!
`http://rextester.com/`

- variables
- comments
- the void keyword
- Arrays
- printf function

CONDITION STATEMENT

- New keywords: **if** and **else**

```
if (<condition>) {
  // expressions to execute when condition is true
} else {
  // expressions to execute when condition is false
}
```

- `<condition>` is an expression that evaluates to an integer value: 0 is **false**, anything else is **true**. For example `a == 2`.

LOOP STATEMENT

- Keyword: **while**

```
while (<condition>) {
  // only executed if condition verified
  // executed infinitely until condition is not verified
}
```

- Keyword: **for**

```
for (<initialization> ; <condition> ; <update>) {
  // only executed if condition verified
  // executed infinitely until condition is not verified
}
```

- **How could you express a `while` loop with a `for` loop and vice-versa?**

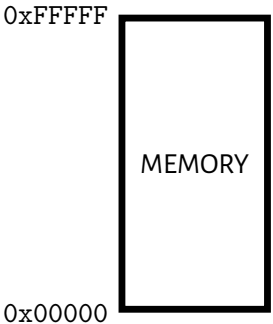
LIVE DEMO

- Playing with loops

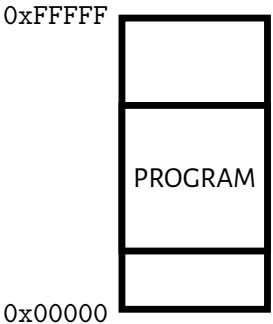
A PAUSE…

- What we've seen so far (condition statements and loop statements) is about *control-flow* .
- All are high level structure, already seen in existing imperative languages.
- So what makes C so **interesting** ?
- ***Low-level features*** such as pointers!

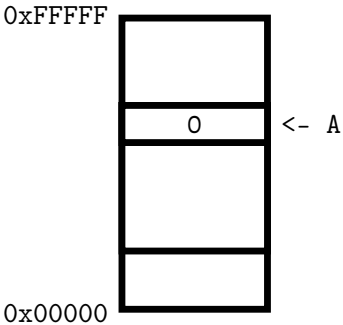
DATA LAYOUT



DATA LAYOUT

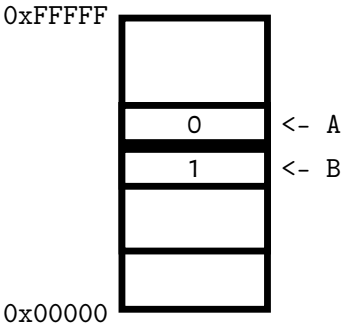


DATA LAYOUT



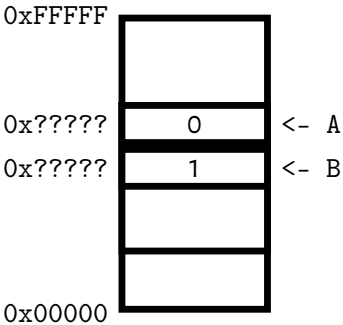

```
int A = 0;
```

DATA LAYOUT




```
int A = 0;
int B = 1;
```

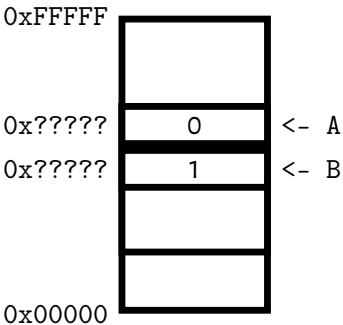
DATA LAYOUT




```
int A = 0;
int B = 1;
```

What are the adresses of A and B in memory?

DATA LAYOUT




```
int A = 0;
int B = 1;
```

What are the adresses of A and B in memory?
Knowledge is power!

POINTERS AND MEMORY ACCESSES

- Operators: **&** and *****

POINTERS AND MEMORY ACCESSES

- Operators: **&** and *
- Given variables: `int a = 0;` and `int b = 1;`
- Then `&a` returns the *address* of `a` in memory.

POINTERS AND MEMORY ACCESSES

- Operators: **&** and *
- Given variables: `int a = 0;` and `int b = 1;`
- Then `&a` returns the *address* of `a` in memory.
- We can start comparing those adresses `if (&a < &b) ...`

POINTERS AND MEMORY ACCESSES

- Operators: **&** and *
- Given variables: `int a = 0;` and `int b = 1;`
- Then `&a` returns the *address* of `a` in memory.
- We can start comparing those adresses `if (&a < &b) ...`
- We can also store this adress in a *pointer* : `int *p = &a;`

POINTERS AND MEMORY ACCESSES

- Operators: **&** and *
- Given variables: `int a = 0;` and `int b = 1;`
- Then `&a` returns the *address* of `a` in memory.
- We can start comparing those adresses `if (&a < &b) ...`
- We can also store this adress in a *pointer*: `int *p = &a;`
- Then `p` is the address of `a` in memory, and `*p` returns the current value of `a`.

POINTERS AND MEMORY ACCESSES

- Operators: **&** and *
- Given variables: `int a = 0;` and `int b = 1;`
- Then `&a` returns the *address* of `a` in memory.
- We can start comparing those adresses `if (&a < &b) ...`
- We can also store this adress in a *pointer*: `int *p = &a;`
- Then `p` is the address of `a` in memory, and `*p` returns the current value of `a`.
- If `a` changes, `*p` changes.

POINTERS AND MEMORY ACCESSES

- Operators: **&** and *
- Given variables: `int a = 0;` and `int b = 1;`
- Then `&a` returns the *address* of `a` in memory.
- We can start comparing those adresses `if (&a < &b) ...`
- We can also store this adress in a *pointer* : `int *p = &a;`
- Then `p` is the address of `a` in memory, and `*p` returns the current value of `a`.
- If `a` changes, `*p` changes.
- What is `&p` ?

EXERCISE

You have to implement the `printbin` function, and to write a small report explaining what you've done and how do you know it works. However if you did not succeed after an hour, don't overdo it but write a report about your attempt, what did you want to know, what was blocking ?

```
void printbin ( int V ); /* This is the function prototype */
```

The function `printbin` returns nothing, but use the standard output (using `printf`) to print the binary representation of an integer `V` .

SOLUTION

TOPICS COVERED

- Variables and Assignement Operators ✓ (T. Bailey, Chapter 1 and 2)
- Numeric Data Types and Conversion
- Arrays
- Arithhmetic and bitwise operators ✓ (T. Bailey, Chapter 2 and 12)
- Compilation, flags, and command-line arguments
- 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
- Structures and Unions
- Memory management and segmentation
- Basic libraries
- Makefile
- Debugging

KEY POINTS

- *source file, portability, modularity*
- *types, variables, identifier, initialized, value*
- *function body, expressions, keyword, return value*
- *condition statement*
- *loop*
- *control-flow, low-level features*
- *address, pointer*

REFERENCES

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