A first C program using Code::Blocks

CSE 1310 – Introduction to Computers and Programming
Alexandra Stefan
University of Texas at Arlington

Outline

- Program: write, compile, run/execute, input, output
- 3 types of errors
- Program syntax rules and error messages
- Simple program output with *printf*
- Starting code template
- Review and list of terms

Writing and running a program

- Writing vs running a program
  - Write a recipe vs cooking according to a recipe
  - Installing a program/game/app vs lunching that game
- Program output
  - what your program "sends out" while it runs
  - Most common form: "print to the screen"
- Program input
  - What data/information is sent into the program while it runs.
  - Most common form: from the keyboard

Errors

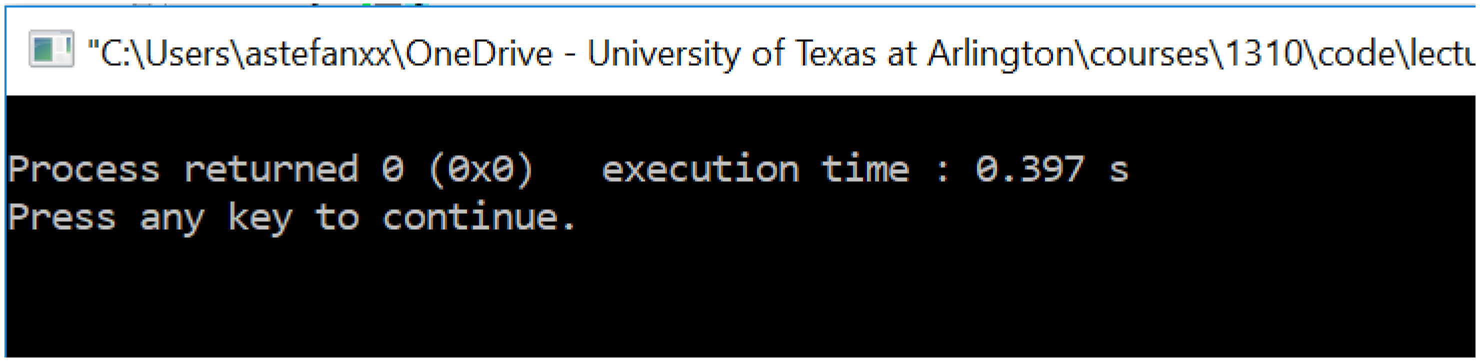
1. *Syntax errors*
   - You do not follow the rules for writing code
   - The compiler catches them and
   - Reports an error message
   - You cannot run the code until you fix it
2. *Logical errors (at runtime)*
   - the program runs, but does the wrong thing (e.g. incorrect computations). E.g. prints: Hello *Wold*
3. *Program crashes (at runtime)*
   - Program “crashes/fails/aborts” during execution.
   - E.g. Give input of the wrong type (e.g. text instead of number)

Create and run a C file in Code::Blocks

- Create a C file with the content below.
  - If you do not remember how to do that, see the Slides on Installing Code::Blocks.

```
int main(void)
{
    return 0;  // This is the last instruction that will execute
}
```

- You should be able to build and run it and that will bring up the execution window



"C:\Users\astefanxx\OneDrive - University of Texas at Arlington\courses\1310\code\lectu

```
Process returned 0 (0x0)   execution time : 0.397 s
Press any key to continue.
```

Create a new C file continued

- Once you have created a C file you can:
  - See that file from File Explorer
  - Open it in the Code::Blocks Editor window
  - Open it with ANY other editor (e.g. Notepad++). You can also edit it in Notepad++ and then open it, build and run it in Code::Blocks
  - Compile and run the program with the "Build and run" button:

  or build it separately and then run it but you risk executing a previous version of the program (not the most recent one) so **ALWAYS use "Build and run"**

  - After you build anyou can see that 2 other files were created: welcome.o (called "object file") and welcomed run.exe (called "executable file")

  - *The .c (i.e. welcome.c) file is called the "**source file**" or the "**source code**" what you will be submitting for your homework. Do NOT submit the .o or .exe files that are created after compiling*

Source code (or *source file*) vs *executable file* (*EXTRA material*)

- The executable file, .exe, (e.g. welcome.exe) can run on its own even on other machines (that match the processor instruction set, operating system and have the libraries the program depends on) even if there is no C compiler on that machine. If you open a command-line window you can run the .exe (e.g. welcome.exe) file by itself (from outside of Code::Blocks). Try these:
  - Double click the .exe file
  - Run .exe from command window
    - cd to directory with welcome.exe
    - Run the code by just typing: welcome.exe
  - Delete or rename welcome.c and try to run welcome.exe again (it works)

- You can also compile code from the command window, but you first need to setup the system variables to include the path to the C compiler.

- If the source code (.c) respects certain standards/rules, it can be compiled on any machine that has a compiler that respects the same standards.
- This content is for deepening your understanding of how C works.

Syntax rules

- The simplest C program is:

```
int main(void)
{
    return 0;  // This is the last instruction that will execute
}
```

- Every C program we want to run on its own must have a `main` function.
- Execution starts at `main`
- Must match {} and ()
- ; - Almost every line of C code must end with ; (we will see some exceptions)
- When clicking “Build and run” the code is compiled and then it is executed (runs). We see a black window. That is the execution window. For more complex programs it will display the program output and will take input from the user.
- Syntax/syntax rules – are the rules that define a correct C program.
    - Syntax error - If the syntax rules are not respected the code will not compile. We call that a syntax error. E.g. Try removing the ;
    - Meeting the syntax rules is the minimum requirement of a program.
    - If a program has no syntax error (it compiles and runs) it may still be wrong (e.g. not compute the correct result or abort before finishing its work)
- Keywords – special words that the compiler knows: `int, void, return`
- The program is CASE SENSITIVE – you must use the exact names for keywords, the main function and other functions you call, identifiers, etc

Syntax rules - spaces

```
int main(void)
{
    return 0;  // This is the last instruction to execute
}
```

- Tokens - smallest building components of a program:
  - Key words, specific symbols (,;, (,) , [,]), literals (values such as 0), identifiers (variable names and function names)
- Spaces
  - At least one space is needed to separate consecutive keywords and identifiers.
    - E.g. `return 0`   `int main`   (try with no space: `return0` `intmain`
  - Otherwise, any number of spaces (or no spaces) is fine
    - Modify spaces and see if the program still compiles (has no syntax error)
  - Spaces in strings are respected: **"CSE1310"** vs **"CSE 1310"** vs **"CSE   1310"**
- Indentation – is not required syntax, but it makes the program readable. You must use it. (It is a CSE1310 learning outcome)
- New lines – are not required, but they make the program readable. You must use them.
- Practice with it!

C code – syntax rules - comments

```
int main(void)
{
    return 0;  // This is the last instruction that will execute
}
```

- Comments – a way to write notes/information useful to the programmer in understanding/clarifying what the code does. They are IGNORED by the compiler.
- Single line comments: // indicates a "line comment" , meaning that the text *following* it, on the same, line will be a comment (not executed). Code before it, if any, WILL be executed.
- Multiple line comments are indicated by start and end special symbols: /* */ (must use the * and / in the exact order as shown)

E.g.:

```
/*  this
is a comment
on multiple lines */


// This is a comment on one line.
```

Program output

- Program output – content that the program prints in the execution window.
  - Later we will see how to make the program print/write into a text file.
- E.g. We want the program to print: *Welcome to CSE 1310!*

Code

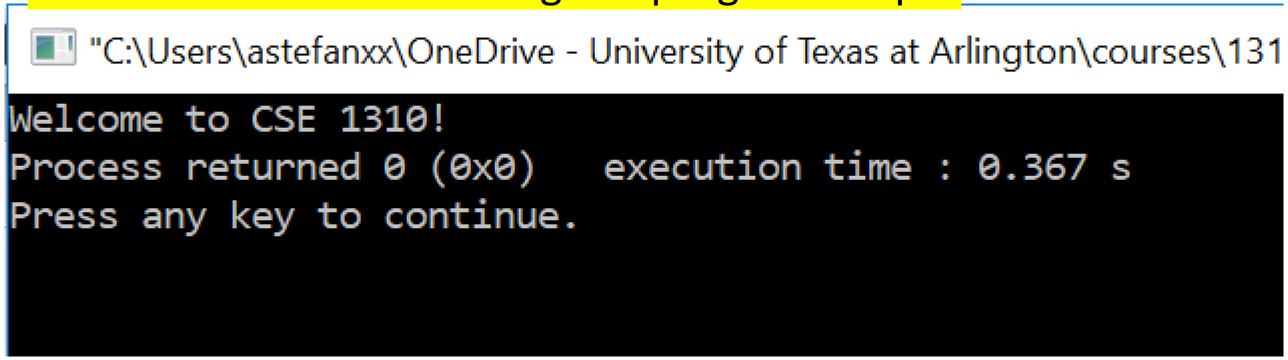
```
#include <stdio.h>

int main(void)
{
    printf("Welcome to CSE 1310!");
    return 0;
}
```

This line is a *preprocessor directive*
It allows your code to use all functions from the stdio.h library.
< > indicate that it is a standard library that came with the compiler (as opposed to one created by you or another programmer).
Learn it by heart.

Calls the *printf* function (from the stdio.h library) and prints the text given in " " . It must have that exact spelling and have () and double quotes " " as shown.
We'll learn to do more with it soon.

Execution window **showing the program output**



"C:\Users\astefanxx\OneDrive - University of Texas at Arlington\courses\131

```
Welcome to CSE 1310!
Process returned 0 (0x0)   execution time : 0.367 s
Press any key to continue.
```

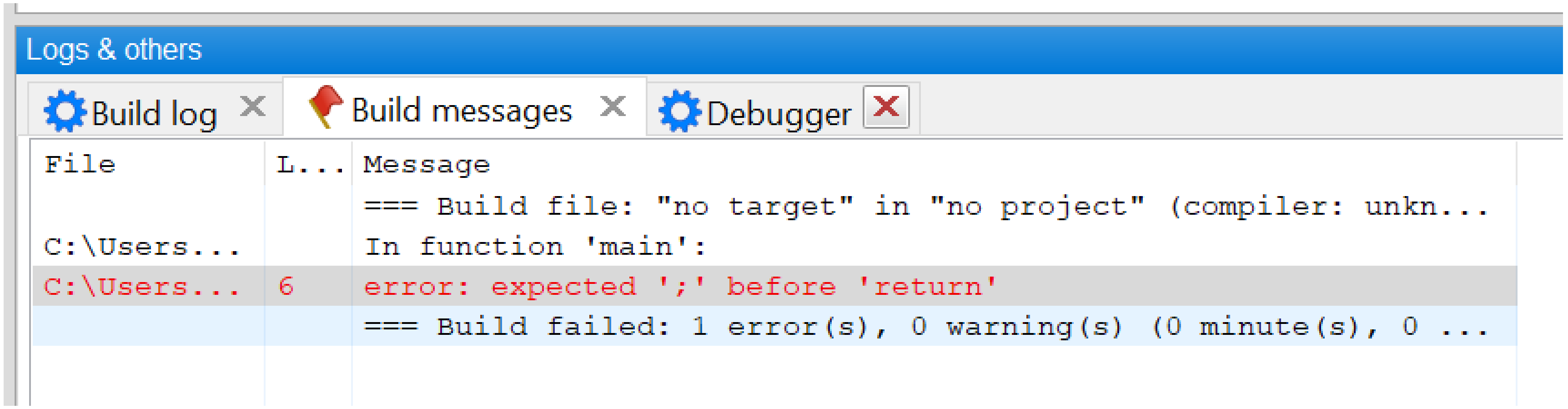
Error messages for Syntax errors

- The error message:

```
error: expected ';' before 'return'
```

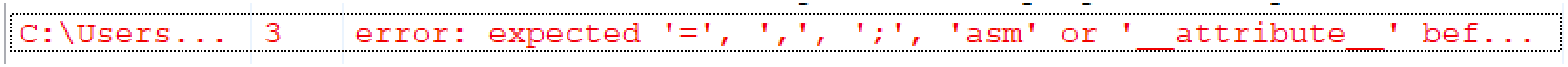
- indicates the problem **';' expected**
- And the number **6** tells you what line of code has that problem

- You will see many error messages. You should not be scared by them, but instead use them to understand what is wrong with your program.



```
File          L...  Message
                    === Build file: "no target" in "no project" (compiler: unkn...
C:\Users...         In function 'main':
C:\Users...   6     error: expected ';' before 'return'
                    === Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 ...
```

- Some errors messages immediately describe the problem and/or the fix, while others do not. See below error from using `int main {` instead of `int main(void) {`



```
C:\Users...   3     error: expected '=', ',', ';', 'asm' or '__attribute__' bef...
```

Explore syntax errors and see error messages

- Try to violate syntax rules and see what happens.
  - If you get error messages, read them, and try to understand them. Work with a partner: they introduce a syntax error and you
    1. First try to understand the problem from the error message.
    2. Fix it.
- Suggested changes:
  - Missing semicolon,
  - Move statements outside main()
  - Mismatch () and/or {}
  - Modify spelling for keywords, *main* or *printf*.
  - Remove required spaces
  - Add extra spaces
  - Remove all the spaces you can so that the program will still be correct.
  - Other

Template for your program:

- You can start every program with this template.
- You should not copy/paste this every time, but type from memory until you learnt it.

```
/* This program does …..
*/

#include <stdio.h>  // need for printing and user input

int main(){
    // Replace this line with your code
    return 0;
}
```

Review and terms

- The 3 types of errors
- Program syntax
  - Matching {}
  - ; at end of statement
- Compile vs run
- Source code vs executable files vs program output
- Keywords: `int, return, include, void`
- Syntax and syntax rules
- Printing function and its syntax:
  - printf(…);
- Program execution
  - Where it starts (first statement in *main)()*)
  - Executes line by line in given order
  - (Later: Use debugger to see this execution)
- Comments:
  - One line
  - Multiple lines

Use debugger to see program execution

- We will do this later