Data Structures in C

Prof. Georg Feil

C Programming Fundamentals

Summer 2018

Acknowledgements

- These lecture slides are partly based on material by Prof. Magdin Stoica and Prof. Simon Hood
- Media tagged ‘CC’ is under Creative Commons license
- Additional sources are cited separately

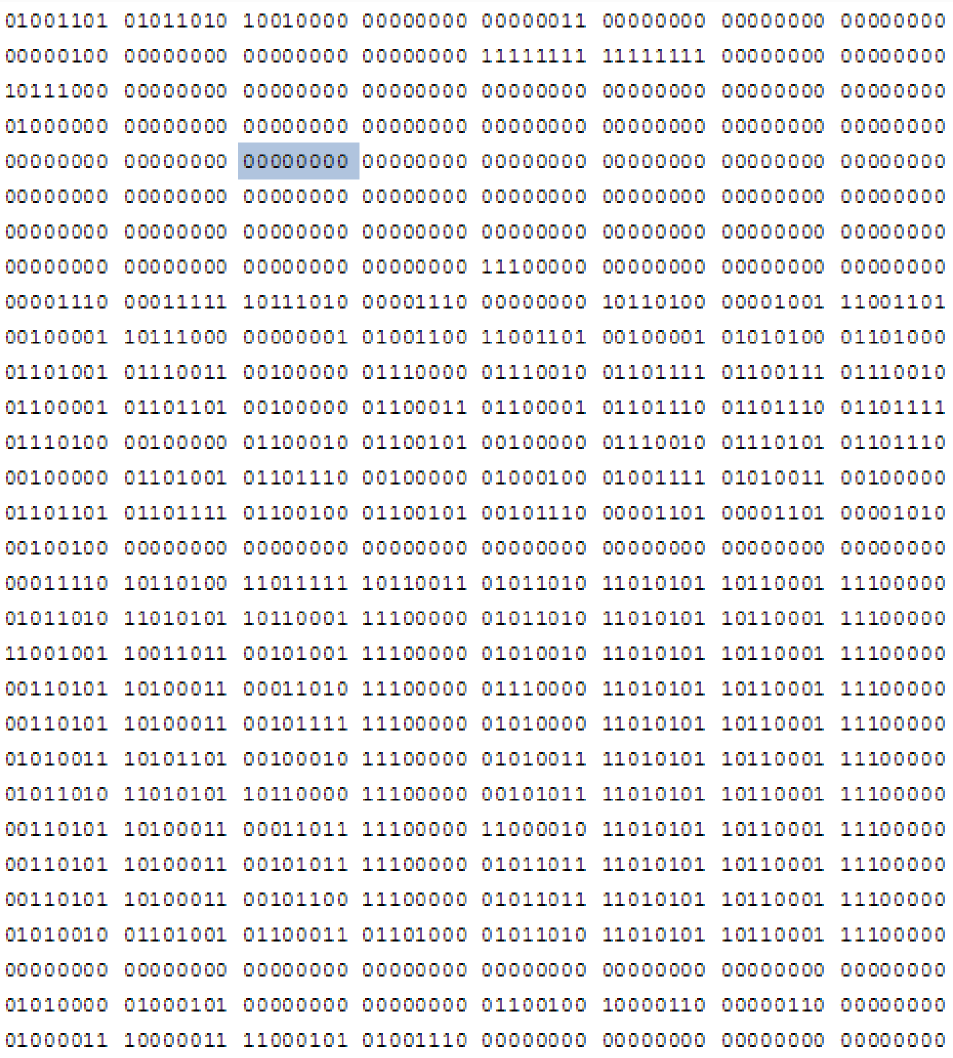
Reading Assignment (required)

- C for Programmers (supplementary textbook)
  - Sections 1.1 – 1.5
  - Sections 2.1 – 2.5

- While viewing these slides you may want to start the software download (see “Exercise 1”)

Programming languages (review)

1st Generation: Machine Language



```
01001101 01011010 10010000 00000000 00000011 00000000 00000000 00000000
00000100 00000000 00000000 00000000 11111111 11111111 00000000 00000000
10111000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 11100000 00000000 00000000 00000000
00001110 00011111 10111010 00001110 00000000 10110100 00001001 11001101
00100001 10111000 00000001 01001100 11001101 00100001 01010100 01101000
01101001 01110011 00100000 01110000 01110010 01101111 01100111 01110010
01100001 01101101 00100000 01100011 01100001 01101110 01101110 01101111
01110100 00100000 01100010 01100101 00100000 01110010 01110101 01101110
00100000 01101001 01101110 00100000 01000100 01001111 01010011 00100000
01101101 01101111 01100100 01100101 00101110 00001101 00001101 00001010
00100100 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00011110 10110100 11011111 10110011 01011010 11010101 10110001 11100000
01011010 11010101 10110001 11100000 01011010 11010101 10110001 11100000
11001001 10011011 00101001 11100000 01010010 11010101 10110001 11100000
00110101 10100011 00011010 11100000 01110000 11010101 10110001 11100000
00110101 10100011 00101111 11100000 01010000 11010101 10110001 11100000
01010011 10101101 00100010 11100000 01010011 11010101 10110001 11100000
01011010 11010101 10110000 11100000 00101011 11010101 10110001 11100000
00110101 10100011 00011011 11100000 11000010 11010101 10110001 11100000
00110101 10100011 00101011 11100000 01011011 11010101 10110001 11100000
00110101 10100011 00101100 11100000 01011011 11010101 10110001 11100000
01010010 01101001 01100011 01101000 01011010 11010101 10110001 11100000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01010000 01000101 00000000 00000000 01100100 10000110 00000110 00000000
01000011 10000011 11000101 01001110 00000000 00000000 00000000 00000000
```

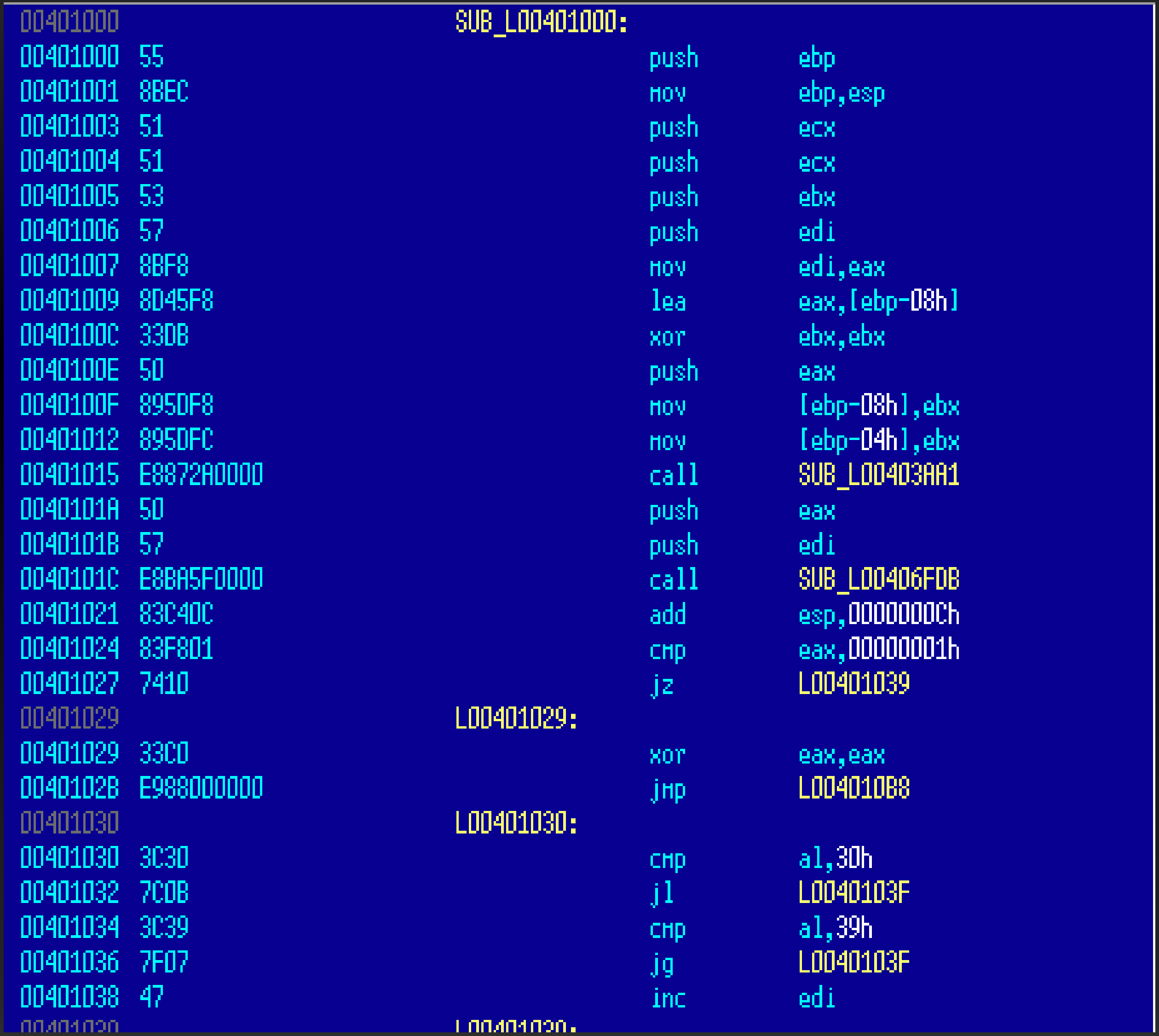
The beginning
of a
.exe file
in binary

An executable (.exe) is a file containing a machine language program (binary 1s and 0s)

```
01010100 01101000
01101001 01110011 00100000
01110000 01110010 01101111
01100111 01110010 01100001
01101101 00100000 01100011
01100001 01101110 01101110
01101111 01110100 00100000
01100010 01100101 00100000
01110010 01110101 01101110
00100000 01101001 01101110
00100000 01000100 01001111
01010011 00100000 01101101
01101111 01100100 01100101
```

Binary for
“This program
cannot be run
in DOS mode”

2nd Generation: Assembly Language



```
00401000                           SUB_L00401000:
00401000  55                                      push        ebp
00401001  8BEC                                    mov         ebp,esp
00401003  51                                      push        ecx
00401004  51                                      push        ecx
00401005  53                                      push        ebx
00401006  57                                      push        edi
00401007  8BF8                                    mov         edi,eax
00401009  8D45F8                                  lea         eax,[ebp-08h]
0040100C  33DB                                    xor         ebx,ebx
0040100E  50                                      push        eax
0040100F  895DF8                                  mov         [ebp-08h],ebx
00401012  895DFC                                  mov         [ebp-04h],ebx
00401015  E8872A0000                              call        SUB_L00403AA1
0040101A  50                                      push        eax
0040101B  57                                      push        edi
0040101C  E8BA5F0000                              call        SUB_L00406FDB
00401021  83C40C                                  add         esp,0000000Ch
00401024  83F801                                  cmp         eax,00000001h
00401027  7410                                    jz          L00401039
00401029                           L00401029:
00401029  33C0                                    xor         eax,eax
0040102B  E988000000                              jmp         L004010B8
00401030                           L00401030:
00401030  3C30                                    cmp         al,30h
00401032  7C0B                                    jl          L0040103F
00401034  3C39                                    cmp         al,39h
00401036  7F07                                    jg          L0040103F
00401038  47                                      inc         edi
00401039                           L00401039:
```

Example of an assembly language program for 32-bit Intel processors

3rd Generation Programming Languages

- Also called high-level languages
- Modern languages
  - Java
  - C and C++
  - C#, Visual Basic
  - Python
  - PHP
- Legacy languages, still in use today
  - COBOL
  - FORTRAN
  - Ada

The C Programming Language

C Language History

- C was developed starting in 1972 to implement unix on the PDP-11 minicomputer
  - Dennis Ritchie was the main designer, he wrote the first C compiler
  - Based on a language called B (Thompson & Ritchie, 1969)

- First book on C was published in 1978, by Brian Kernighan & Dennis Ritchie
  - "The C Programming Language"

- C was updated in 1989, by the American National Standards Institute (ANSI)
  - This standard is often called C89
  - ANSI C added many modern features like *stronger type checking* and *function prototypes*
  - New edition of Kernighan & Ritchie's C book was published

C History (cont'd)

- C was updated again in 1999 by ISO/IEC, and the same standard was also adopted by ANSI
  - This standard is called C99
  - Adds ability to declare variables anywhere in a scope (not just the start), a Boolean data type 'bool', and support for 64-bit integer types
  - Most of the changes were already supported in existing C compilers, and in C++, but not always in the same way
- *We will use the C99 standard version of C in this course*
  - Another update of the C standard came out in 2011 (C11)

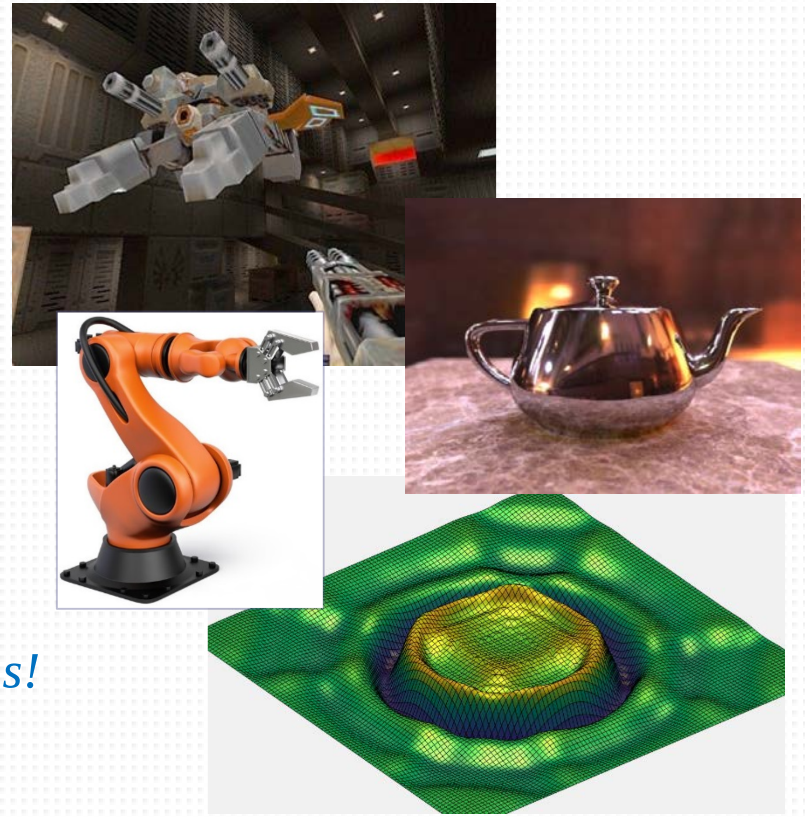
… in parallel with this

- C++ was developed by Bjarne Stroustrup, starting 1985
  - Adds object-oriented features to C (classes and objects)

Where is the C language used?

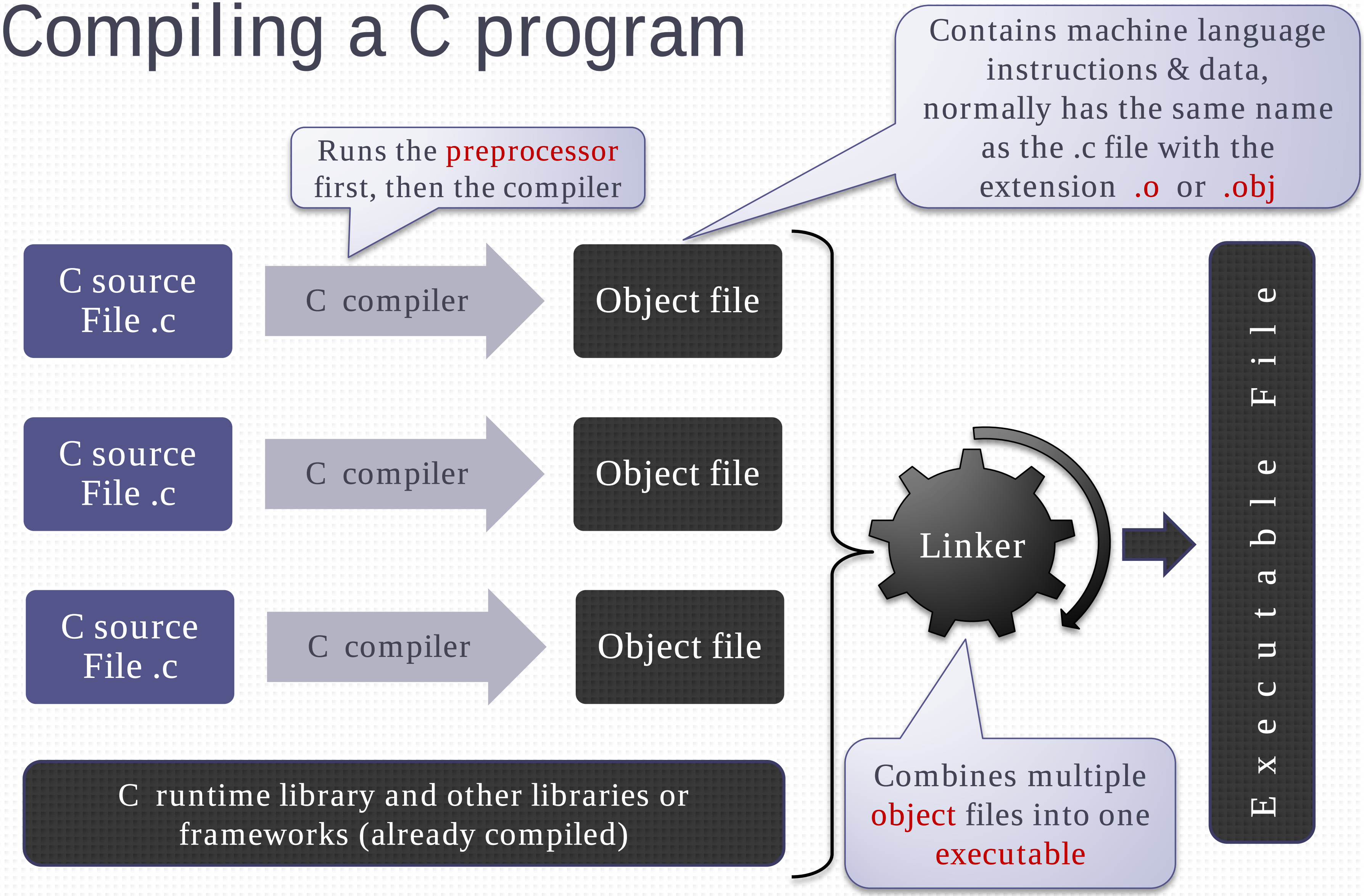
- C is one of the most widely known and used computer languages in the world
  - #1 or #2 in lists of most-popular languages

- C is used for
  - Game programming (C/C++)
  - Advanced graphics, e.g. OpenGL
  - Scientific programming
  - System (OS) programming and device drivers
  - Embedded systems
  - *Creating advanced data structures!*



C is a “true” compiled language

Compiling a C program



Runs the preprocessor first, then the compiler

Contains machine language instructions & data, normally has the same name as the .c file with the extension .o or .obj

Combines multiple object files into one executable

Types of C source files

- There are two main types of C source files:
  - Regular C source files: each file has a **.c** extension. These files contain C statements, functions etc.
  - C header files: each file has a **.h** extension. These files should contain C *declarations* only, no statements
- The compiler processes each .c file individually, and accesses .h files indirectly
  - Header files are normally accessed by including them from .c files, or other .h files with `#include`
- A big program may consist of many **.c** & **.h** files
- Do not use **.cpp** files, these are for C++ programs!
- Do not use spaces in file names!

What does a C program look like?

- A simple C program looks like this:

```
#include <stdio.h>     // one or more includes

// The main function
int main(int argc, char** argv)
{
    printf("C programming is fun!\n");
    return 0;
}

// Other functions…
```

What does a C program look like?

- The main function is similar to the main method in Java
  - It’s where program execution starts

- The #include statement is similar to ‘import’ in Java

- Here it’s used to give us access to input/output functions from the C library like printf, declared in stdio.h
  - #include is like copying/pasting the referenced file at that spot
  - For system .h files use < >, for your own .h files use " "
  - Statements starting with # are not handled by the C compiler, they’re handled by the C preprocessor before compiling

Exercise 1: Installing Dev-C++

- In this course we'll be using Dev-C++
  - An open-source Integrated Development Environment (IDE) for Windows
  - Includes the very popular GNU C compiler, GCC
- GCC can be used from the command line or the graphical IDE
- Can compile C programs and C++ programs
- Get Dev-C++ here (download button is green, in middle of page): http://sourceforge.net/projects/orwelldevcpp/

  (Don't use other download sites, you might get an old version)
- Install, keeping all the default options
  - Install directory should be C:\Program Files (x86)\Dev-Cpp

Exercise 1: Running Dev-C++ First Time

- You can run the Dev-C++ IDE using the desktop shortcut made for you by the installer

- The first time you run Dev-C++ it will ask some “first time configuration” questions
  - You can choose the default answer for all questions (click “Next”)

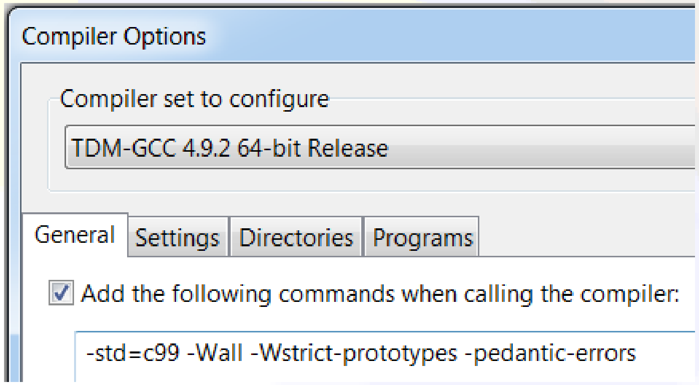
- Note: AVAST anti-virus conflicts with Dev-C++, deletes executables!

Exercise 1: After Installing Dev-C++

- Let’s set some compiler options as we’d like to have them for this course
  - Tell the compiler to use the C99 language standard
  - Output some useful warnings that aren’t shown by default
- Go to: Tools > Compiler Options
- Put this line in the box where it says “Add the following commands when calling compiler” and check the checkbox

```
-std=c99 -Wall -Wstrict-prototypes -pedantic-errors
```

- This means use standard C99 and display some useful warnings
- Also enable debugging:
  Settings tab > Linker tab >
  Generate debugging info (-g3) → Yes



Compiler Options

Compiler set to configure

TDM-GCC 4.9.2 64-bit Release

☑ Add the following commands when calling the compiler:

```
-std=c99 -Wall -Wstrict-prototypes -pedantic-errors
```

Exercise 2: Hello World

Now let’s try a simple program…

- For this exercise we’ll enter, compile, and run the “Hello World” program in C
- You also might get some experience debugging C programs…. if you make a typing mistake by accident. Or on purpose… try to see what some errors do.

1. Start the Dev-C++ IDE
2. Choose File > New > Source File
3. Enter the program on the next slide

Exercise 2: Hello World (cont’d)

- Fill in your own name where indicated
- Watch out for upper/lower case
  - C is case-sensitive, like Java
- Save it with the name hello.c

Remember don’t use extension .cpp Also choose a location with no spaces anywhere in the path name.

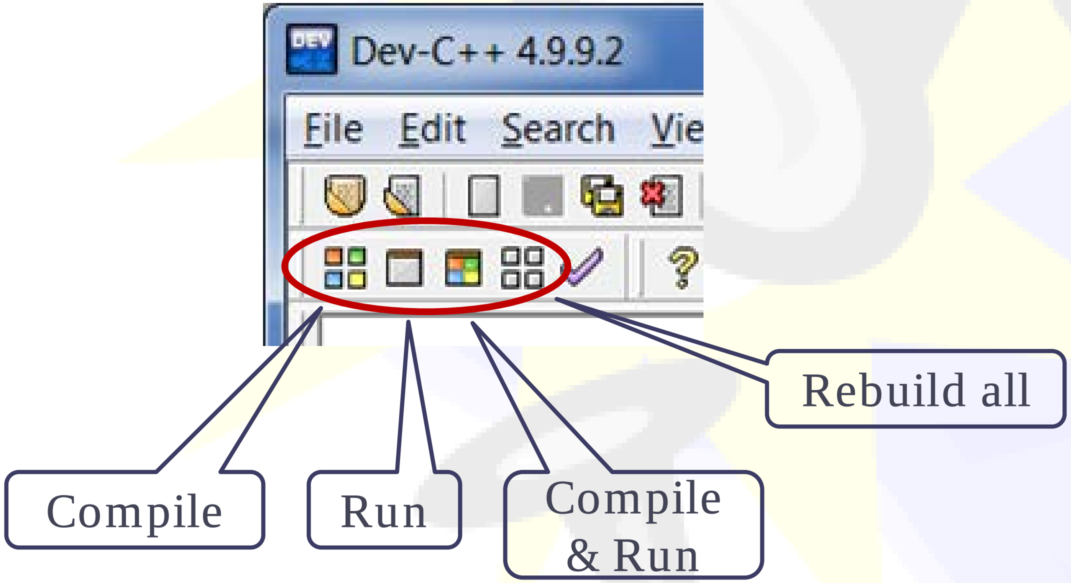
```
#include <stdio.h>

/* This is our first program in C */
int main(int argc, char** argv)
{
    printf("Hello, my name is [your name].\n");
    printf("I hope this C program works!\n");

    return 0;
}
```

Exercise 2: Hello World (cont'd)

- Compile and run it using buttons in the IDE



Compile

Run

Compile & Run

Rebuild all

Looking up C Library Documentation

- To learn about useful functions in the C library, use this site: www.gnu.org/software/libc/manual/

- To find information on a specific C library function
  - Click on HTML - one web page per node
  - Go to Appendix B: Summary of Library Facilities
  - Search (ctrl F) for the name of the function
    - Note it may be quicker for some functions to search for *name (*

- Do this now to look up the documentation for printf
  - For more related information see Formatted Output

Good Standards and Organization

- You should create a directory (folder) for our course to keep things organized
  - You will definitely need sub-directories inside it as well so your assignments, exercises, etc. are organized
  - C doesn’t force you to use certain file or directory names like Java
- You should follow good programming practices and standards when writing code – See our Coding Standards on SLATE under General
  - Proper indenting and code formatting
  - Writing good comments
  - Good choice of variable and function names, etc.

Exercise 3: Hello World++

- Change the Hello World program to print your name 10000 times using a loop
  - Hint: Loops in C work just like Java!
- Save it with the name hello2.c
- To keep printing on the same line take away the ‘\n’ in printf

Steps to develop a C program

Basic steps working in C

1. Plan and design your program

2. Edit your program ***source file****(s)*
   - C source files are text files

3. Compile your program using a ***compiler*** and ***linker***

4. Run your program and test it

Step 1: Plan and Design

a) Understand the customer's needs

- The customer will usually tell you, if not ask
- In this course your customer is your professor!

b) Understand how the user will use the program (use cases)

- Remember: you are not the "end" user

c) Establish the software requirements

- This can take a long time and a lot of work for a big system

d) Analyse the requirements to choose software platforms and high-level architecture

- What computer hardware? What language and tools? How many modules? Which software libraries or frameworks will be used?

e) Perform the detailed design of software modules, functions, and data structures (based on requirements)

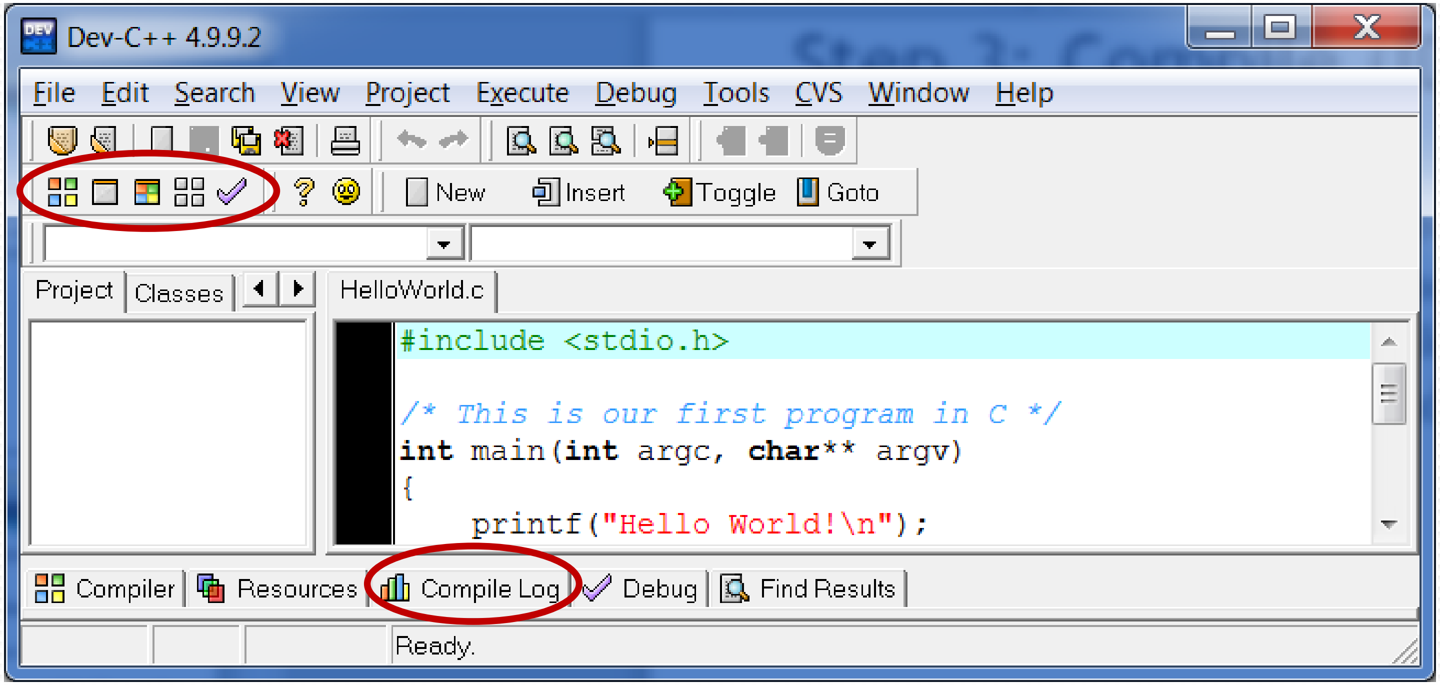
Step 2: Edit Source Code

- C source files are text files, so you can use any text editor to create/modify them
  - Notepad or Wordpad [Windows]
  - vi, emacs, gedit etc. [Unix/Linux]

- All Integrated Development Environments include text editors, e.g.
  - Dev-C++ [Windows]
  - Microsoft Visual C++ [Windows]
  - Eclipse with CDT [Unix/Linux, Mac OS X, Windows]
  - NetBeans [Unix/Linux, Mac OS X, Windows]

Step 3: Compile (IDE)

- When you just press a button to compile
  - The IDE runs the compiler and linker for you (in our case GCC)
  - The output is an executable file (if you have no errors)
- You can see what commands were used by clicking on the Compile Log tab



```
#include <stdio.h>

/* This is our first program in C */
int main(int argc, char** argv)
{
    printf("Hello World!\n");
```

Compile and Run buttons

Step 3: Compile (command line)

- You can run the compiler/linker yourself from the command line:

  `gcc [options] file1.c file2.c …`

- (Must set the PATH environment variable correctly)
- To choose the name of your executable file use the gcc -o (output) option, e.g. `gcc prog.c -o MyProgram`
  - If you don't choose a name it will be called a.exe
  - The -o option can go before or after the .c file name(s)
  - There are many other options, you should use `-std=c99 –Wall –Wstrict-prototypes -pedantic-errors`
  - Note that we never try to compile .h files

Step 4: Run and test your program

- The compiler automatically starts the linker which produces a single executable file
  - Has the file extension **.exe** on Windows
  - Sometimes called a binary file
- Use the Dev-C++ “run” button to run the program
  - Or type the .exe file name in a command window
- Note that we’ll be writing C console programs
  - No graphics or Graphical User Interface (GUI), only text
  - These programs always run in a command window on your PC

MCQ

Which of the following best describes what the linker does when building a C program with more than one .c file?

a) Combines several source files to produce one executable file.

b) Compiles all the source files to produce one object file.

c) Combines several object files to produce one executable file.

d) Processes lines that start with #, like #include, to produce one object file.

Data Structures and Algorithms

Data Structures and Algorithms

- A data structure stores data in a computer so that it can be used efficiently
  - We may need to store and work with large amounts of data
- An algorithm is a step-by-step sequence of instructions for solving a problem
  - Algorithms work with data structures!
- Data structures and algorithms are inseparable aspects of computing
  - The 1st defines how data is stored, the 2nd defines how data is used

Examples of Data Structures

- Array
- Dynamic Array
- Stack
- Queue / buffer
- Linked List
- Tree
- Graph
- Hash Table

Hint: You’ve already written programs with the first three!

Efficiency

- We all want efficient and effective computers and computer programs
- It all comes down to resources
  - Ideally, our algorithms and data structures should minimize their use of system resources like processor cycles & memory
- A large part of this course will be about writing optimized, well-structured code, as opposed to code that works but is wasteful or disorganized

Organizing data wisely facilitates efficient algorithms

Some Differences between C and Java

C	Java
Language standard issued cooperatively by ANSI (U.S.) and ISO/IEC (worldwide)	Not internationally standardized
Programs compile to machine language and run on the real machine (*fast*, able to access hardware directly)	Programs compile to bytecode and run on a virtual machine (JVM), can't access h/w directly
C Library for useful functions related to input/output, string manipulation, math	Java Library, similar purpose but looks quite different (OO)
C compilation has two additional steps: - Preprocessor to handle statements that start with #, like #include (runs first) - Linker combines multiple object files	Java compiler compiles, then the Java Virtual Machine interprets bytecode and may do Just In Time compilation
Source file names are flexible, files can be located in any folder you choose	Source file/folder names must be the same as the class/package
Memory management done by programmer	Memory management is automatic

Some Similarities between C and Java

C	Java
Is a 3rd generation language	Is a 3rd generation language
Source files are plain text (.c, .h)	Source files are plain text (.java)
C compiler can give errors or warnings	Java compiler can give errors or warnings
You can compile and run from the command line or an IDE	You can compile and run from the command line or an IDE
Uses { } for blocks/scopes, and ; for end of statements	Uses { } for blocks/scopes, and ; for end of statements
Program starts in a function called main	Program starts in a method called main
I know how to write the Hello World program in C!	I know how to write the Hello World program in Java!