KU LEUVEN

C: an introduction

Basic workflow
A first program

Creating a C Program

- Have an idea about what to program
- Write the source code using an editor
- Compile the source code and link the program using a C compiler
  - Fix errors, if any
- Run the program and test it
  - Fix bugs, if any

https://www.tacc.utexas.edu/c/document_library/get_file?uuid=6041435f-cda4-442d-94ef-15a73b32387d&groupId=13601

Your computing environment

KU LEUVEN

Bare necessities

- Minimal requirements:
  - Editor: edit source code files
  - C compiler (& debugger)
- Considerations
  - OS: Windows / Linux – Mac
  - Command line environment / (graphical) Integrated Development Environment
- See also: https://en.wikibooks.org/wiki/C_Programming/What_you_need_before_you_can_learn

KU LEUVEN

Windows

- Visual Studio
  - Full IDE
  - Command line
- Windows Subsystem for Linux (WSL)
  - Work completely in linux environment – command line
- Mix IDE (windows) and compiler (linux) – Visual Studio Code

First program

Form of a C program

- Basic building block: function
  - Every C program consists of one or more functions.
    - One of these functions must be called `main`.
    - Execution of the program always begins by carrying out the instructions contained in `main`.

```
main()
{}
```

    - *File: minimal.c*

A traditional first example

```
#include <stdio.h>

int main()
{
printf("Hello this is C \n");
return 0;
}
```

Preprocessor directives to include header files

Execution begins by calling `main` ends when `main` returns

Statements are terminated by ;

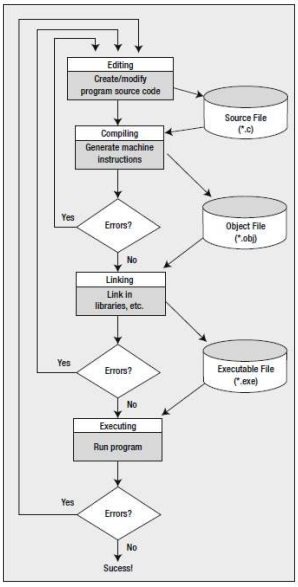
Functions are delimited by curly braces { }

File: hello_c.c

Stepping stones

- Save your program (source code) with ‘c’ extension.
  Example: hello_c.c
- Compile and Link the code (by default, GCC automatically links) `gcc -o hello_c hello_c.c`
- Run the program
  `./hello_c`
- Repeat the steps above every time you fix an error!
- Run program by executing executable
  - No need to have the source code
  - No need for compiler for execution of executable

I.Horton, “Beginning C 5th Ed.”, Apress, 2013



Compiler vs interpreter

- Compiled languages (with static data types) such as C, Fortran, C++:
  - execute very fast | tool of choice if speed is primary concern
  - don't need source code to execute (there may be commercial interest in not revealing the source code)
  - more complicated to write, debug and read code
- Interpreted languages (with dynamic data types) such as Python, Matlab
  - writing code is easier (saves a lot of time)
  - execution of code is slower (can differ by factor 10-1000 from compiled languages)
  - cannot execute without source code
- Source: http://www.southampton.ac.uk/~feeg6002/lecturenotes.html

Hands-on

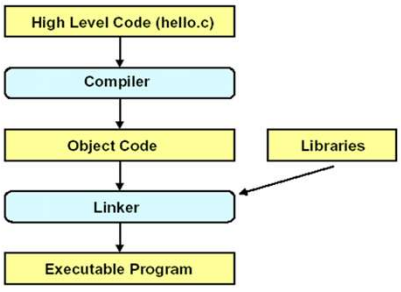
- Write Hello World program (*File: hello_world.c*)

```
#include <stdio.h>
int main() {
printf("hello world!\n");
return 0;
}
```

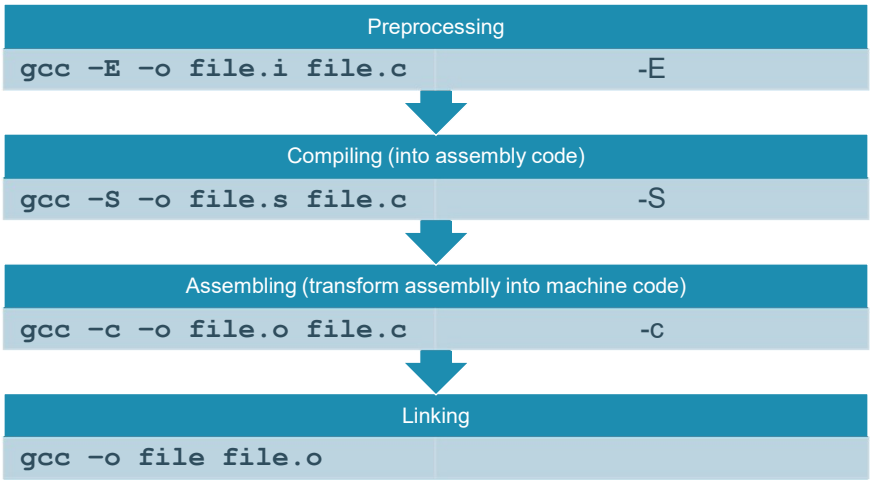
- Compile it
- Execute it
- Extra
  - Remove \n
  - Add another line of text

C: from high-level code to executable

1. Program source code written in C (text file with ‘c’ extension)
2. preprocessor: preprocessing before compilation can start
3. compiler translates into object file
4. linker combines object code with predefined routines from libraries and produces the executable program



Compilation steps



https://www.calleerlandsson.com/the-four-stages-of-compiling-a-c-program/

KU LEUVEN

Compiler options

-E	Pre-process only. Output pre-processed code.
-S	Compile only; output assembly code.
-c	Compile or assemble the source files, but do not link. The output is object files corresponding to each source file.
-o	Instructs gcc to create the output with the specified filename. The default executable is a.out
-save-temps	Saves the intermediate files in the current directory
-Wall	Show all warnings. An essential aid in detecting possible problems when programming in C File: compile_wall.c
-g	Produce debug information, necessary for debugging.
-llibrary	Links to a standard library. Use -lm to load the standard math library.
-Dmacro	Define a macro, one can also use -Dmacro=val in order to assign a value for the macro. This will be used for preprocessing all files.

KU LEUVEN

Hands-on

- Use the Hello World program (*File: hello_world.c*)

```
#include <stdio.h>
int main() {
printf("hello world!\n");
return 0;
}
```

- Use the different gcc options and check the generated output

See also

- Compiling: https://www.thegeekstuff.com/2011/10/c-program-to-an-executable/
- Linking: https://www.thegeekstuff.com/2011/10/gcc-linking/
- Overview: https://www.calleerlandsson.com/the-four-stages-of-compiling-a-c-program/

comment

- Multi-line comment **`/*  */`**

```
/* this is a comment
over different lines */
```

- Single line comment start with **`// (c99)`**

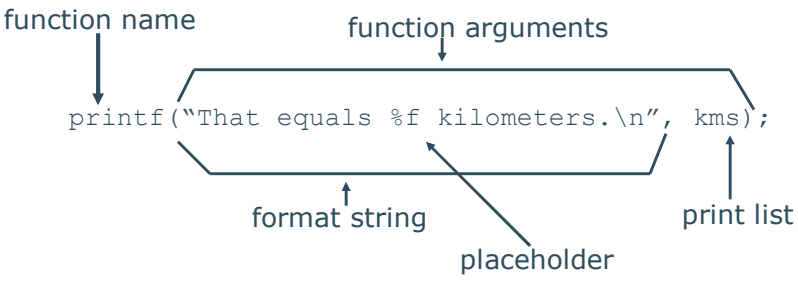
Hands-on

- Write Hello World program (*File: hello_c_comment.c*)
- Put extra comment lines into the code

Basic I/O

- Input/Output Operations and Functions
  - special program units that perform all input/output operations
  - `printf` Function
    - From the standard library, you have to include the `<stdio.h>` header file
    - Output operation - results can be displayed
  - `scanf` Function
    - Input operation - data transfer from outside into computer
- *Files:*
  - *simple_sum.c*
  - *simple_sum_input.c*

The `printf` Function



Common `printf( )` Conversions

specifier	
`%d`	the int argument is printed as a decimal number
`%u`	the int argument is printed as an unsigned number
`%s`	prints characters from the string until '\0' is seen or the number of characters in the (optional) precision have been printed
`%f`	the double argument is printed as a floating point number
`%x, %X`	the int argument is printed as a hexadecimal number (without the usual leading "0x")
`%c`	the int argument is printed as a single character
`%p`	the pointer argument is printed (implementation dependent)

`printf( )` conversions

- Conversions specifications begin with % and end with a conversion character.
- Between the % and the conversion character some extra specifications can:
  - A minus sign specifying left-justification
  - The minimum field width
  - A period separating the field width and precision
  - The precision that specifies
    - Maximum characters for a string
    - Number of digits after the decimal for a floating point
    - Minimum number of digits for an integer

printf( ) Examples

```
int main()
{
   char  ch = 'A';
   char  str[20] = "fresh2refresh.com";
   float flt = 10.234;
   int no = 150;
   double dbl = 20.123456;

   printf("Character is %c \n", ch);
   printf("String is %s \n" , str);
   printf("Float value is %f \n", flt);
   printf("Integer value is %d\n" , no);

   printf("Double value is %f \n", dbl);
   printf("Double value is %lf \n", dbl);
   printf("Double value is %7.2f \n", dbl);
   printf("Double value is %-7.2f \n", dbl);
   printf("Double value is %7.5f \n", dbl);
   printf("Octal value is %o \n", no);
   printf("Hexadecimal value is %x \n", no);
}
```

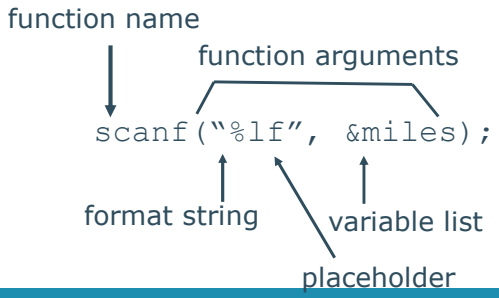
- *File: demo_printf.c*

Special characters

- The backslash \ character in C has a special meaning: escape character
- escape sequences:
  - \n New line
  - \t Tab
  - \b Backspace
  - \r Carriage return
  - \f Form feed (new page)
  - \\ \
  - \" ”
  - \’ ’

The scanf Function

- The placeholder type tells the function what kind of data to store into variable miles.
- The & is the C address of operator. The & operator in front of variable `miles` tells the scanf function the location of variable miles in memory.



http://faculty.kfupm.edu.sa/coe/aimane/ics103

Reading data

- For two variables A and B, both of type double:

```
scanf("%lf %lf", &A, &B);
```

- Format Specifiers

Format specifier	
d	int
u	unsigned int
f	float
e	float (exponential form)
G	float (general form)
lf	double
le	double (exponential form)
lg	double (general form)

- & represents the address of the variable in memory (a pointer reference operator).
- *File: demo_scanf.c*