Podstawy Programowania

Stanisław Gepner

room 117,
stanislaw.gepner@pw.edu.pl, sgepner@meil.pw.edu.pl

Resources

- Internet
- C / C++ programming tutorials
- Stack Overflow
- C / C++ reference online
- Google, Bing, Duck Duck Go ...
- Any Visual Studio or other IDE with C/C++ support
- Online compilers `https://godbolt.org/`
- It is like push-ups, you need practice, not books!

Programming

- Act of writing instructions enabling the computer to perform desired tasks
- Ada Lovelace (1815 - 1852) - The first programmer
- Or was it Charles Babbage (1791-1871)?

Source: Wikipedia, Science Museum / Science & Society Picture Library

C language

- Dennis Ritchie – AT&T Bell Laboratories - 1972
- The C Programming Language - first specification - 1978
- 1989: ANSI C89, 1990: ISO C90
- 1999: C99 standard
- Still in use, and here to stay for a while
  - Wide range of applications. OS, microcontrollers, ATM systems ...
  - Efficiency and performance
  - Provides low level access
  - Influenced C++, Obj. C, C#, Java, ...
- Many of the mannerisms of C have been adapted by many descendent languages. == to test for equality. && for boolean AND, and & for bitwise AND. « and » for bit shifting. % for modular arithmetic. Braces for code blocks. Now these seem almost universal.

First program

- Text file with **.c** extension (or .cpp ...)
- Edit with your favourite text editor (or an IDE)
- // this is a single line comment, this will not be processed
- /* this is a multi line comment, will not be processed */

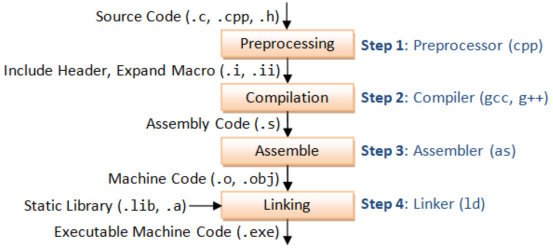
```c
/* Start with a short description of the program */
#include <stdio.h> //Append the I/O library

int main()
{
  printf(" Hello students!! \n ");
}
```

- # lines starting with a hash are preprocesor directives. We will be seeing them.
- There has to be exactly one main function!
- Watch brackets!!
- (Almost) all instructions end with a semicolon ;

Compile and Run

```
$ gcc hello.c
```



Source Code (.c, .cpp, .h)
Preprocessing
Step 1: Preprocessor (cpp)
Include Header, Expand Macro (.i, .ii)
Compilation
Step 2: Compiler (gcc, g++)
Assembly Code (.s)
Assemble
Step 3: Assembler (as)
Machine Code (.o, .obj)
Static Library (.lib, .a)
Linking
Step 4: Linker (ld)
Executable Machine Code (.exe)

In the end a.out

```
$ gcc -Wall hello.c -o hello.out
```

Can get pretty long ...

Basic C elements
Allowable characters

Characters we can use:

- a-z - lower case letters
- A-Z - upper case letters - different!
- 0-9 - digits
- (), [], {} - brackets
- +, -, *, /, % - operations
- !, <, =, >
- &, @, ., ,, :, ;, ', ", #,
- And than some more!

Basic C elements
Keywords

Reserved keywords that have a special meaning to the compiler:

auto	double	int	struct
breach	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Basic C elements

User defined names

As programmers we will need to give names to elements of our program:

- Functions, variables, constants (and some others) need an identifier
- Any sequence of letters, digits and an underscore can be a name
- First character must be a letter or an underscore
- An identifier can not be a keyword (see the previous slide)
- ax123(), x1(), dominateTheWorld(), RuleGalaxy111() are examples of admissible identifiers (names)
- Customary to start with a lower case letter

Comments - ignored by the compiler

Use comments:

- to explain the code
- to describe what is done
- as notes
- for fun

```
/* This is a multi-line
   comment.
   It can go on
   through many lines until this: */

// This is a single line comment, it stretches until the end of line.
     If there is no "new line" and the line is folded, it is still a
     single line.

// sometimes I believe compiler ignores all my comments
```

Comments Real life example

Comment found in one of the Stack Overflow as a warning ...

```
//
// Dear maintainer:
//
// Once you are done trying to 'optimize' this routine,
// and have realized what a terrible mistake that was,
// please increment the following counter as a warning
// to the next guy:
//
// total_hours_wasted_here = 42
//
```

Program structure

```
/* Description - not mandatory but polite*/
#include <stdio.h>
#define PI 4.0*atan(1.0)
```

Program structure

```
/* Description - not mandatory but polite*/
#include <stdio.h> //Preprocessor commands starting with a \#
#define PI 4.0*atan(1.0) // Note no semicolons ";", why?

int main()
{

}
```

Program structure

```
/* Description - not mandatory but polite*/
#include <stdio.h> //Preprocessor commands starting with a \#
#define PI 4.0*atan(1.0) // Note no semicolons ";", why?

int main() //The main function must be there
{ // <- the opening bracket
  // Body of a function, "the meat"
} // <- The closing bracket

// ==== Above obligatory, below additional user defined functions
     ====

int sum_ints(int a, int b)
{
  return a+b;
}
```

Program structure

```
/* Description - not mandatory but polite*/
#include <stdio.h> //Preprocessor commands starting with a \#
#define PI 4.0*atan(1.0) // Note no semicolons ";", why?

int sum_ints(int a, int b); //Function prototypes, a promise to the
     compiler
int a=5; // Definition of global variables if needed. Not mandatory,
     more later.

int main() //The main function must be there
{ // <- the opening bracket
  // Body of a function, "the meat"
} // <- The closing bracket

// ==== Above obligatory, below additional user defined functions
     ====

int sum_ints(int a, int b)
{
  return a+b;
}

....
```

Headers

```
/* Description - not mandatory but polite*/
#include <stdio.h>
#include "myheader.h"
```

- Header files contain constants, functions, other declarations
- System or user generated
- #include <stdio.h> - read the contents of the header file stdio.h
- stdio.h: standard input/output for console and files
- #include <stdio.h> - look for system headers
- #include "mygreatheader.h" - look for user generated headers in ./

Functions

```c

int sum_ints(int a, int b); //A prototype end with semicolon

//type name (arguments)
int main(void)// main can have arguments
{
  return 0; // main is special!
}

//this function is of integer type
int sum_ints(int a, int b) //it accepts two arguments of integer type
{
  return a+b; // since it has a type it must have a return.
}
```