

Computer Science I

Stanisław Gepner

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

Rules and regulations

1. This class is divided into lectures and obligatory laboratory classes.
2. Grading is done using a 0 to 100 point scale.
 - a. 60 points for 2 lecture tests worth 30 points each
 - b. 40 points for laboratories.
 - i. Laboratory grading is to be established by the lab teacher.
3. A passing mark is given only, if student obtained a minimum number of points, i.e.:
 - a. 21 points from the laboratories,
 - b. 31 points from both lecture tests

Failing to achieve the minimum number of points results in a failing grade (2).
4. In case a minimum requirements, outlined in point 3, are passed, the final grade is calculated as follows:
 - a. 0-50: 2.0
 - b. 51-60: 3.0
 - c. 61-70: 3.5
 - d. 71-80: 4.0
 - e. 81-90: 4.5
 - f. 91-100: 5.0
5. The dead line for collecting points is the last day of the semester. After the last day of the semester the grade becomes final. There will be no corrections
6. Laboratory attendance is obligatory. Each absence needs to be excused, e.g. with a medical leave. Failing to do so results in deduction of 3 points.
7. Laboratory groups are formed based on the dean's office list enumeration.
 - a. 1-12 room 120
 - b. 13-24 room A3
 - c. 25 and higher will be assigned to an additional group
8. On the last lecture there will be an additional, correction test worth 50 points.
 - a. If better, the mark of the correction test replaces results of both lecture tests.
 - b. To be eligible for the correction test:
 - i. At least one of the lecture tests must be passed with more than 50% (16 points).
 - ii. Laboratory homework assignment must be passed.
 - iii. Student must be present at at least 8 lectures.

Resources

- Internet
- C / C++ programming tutorials
- Stack Overflow
- C / C++ reference online
- Google, Bing, Duck Duck Go ...
- Visual Studio 2015 Community or Visual Studio 2013
- 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



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, ...

First program

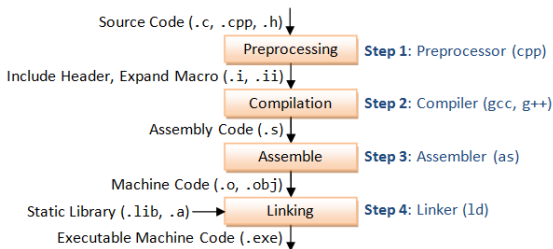
- Text file with `.c` extension (or `.cpp` ...)
- Edit with your favourite text editor (or ...)
- `//` this is a single line comment, this will not be processed
- `/*` this is a multi line comment, will not be processed `*/`

```
1  /* Start with a short description of the program */
2  #include <stdio.h> //Append the I/O library
3
4  int main()
5  {
6      printf("Hello students!!\n");
7  }
```

- `#` lines starting with a hash are preprocessor 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
```



In the end a.out

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

Can get pretty long ...

Basic C elements

Allowable characters

- a-z - lower case letters
- A-Z - upper case letters - different!
- 0-9 - digits
- (), [], - brackets
- +, -, *, /, % - operations
- !, |, =, _
- &, @, ., ,, :, ;, ', ", #,
- There will be more!

Basic C elements

Keywords

auto	double	int	struct
break	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

- Function names, variables, constants ...
- Any sequence of letters, digits and an underscore
- First character must be a letter
- Can not be a keyword
- `ax123()`, `x1()`, `dominateTheWorld()`, `RuleGalaxy111()`, ...
- 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

```
1  /* This is a multi-line
2     comment.
3     It can go on
4     through many lines until this: */
5
6  // This is a single line comment, it stretches until the end of
7     line. If there is no "new line" and the line is folded, it
8     is still a single line.
9
10 // sometimes I believe compiler ignores all my comments
```

Comments Real life example

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

```
1  //  
2  // Dear maintainer:  
3  //  
4  // Once you are done trying to 'optimize' this routine,  
5  // and have realized what a terrible mistake that was,  
6  // please increment the following counter as a warning  
7  // to the next guy:  
8  //  
9  // total_hours_wasted_here = 42  
10 //
```

Program structure

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

Program structure

```
1  /* Description - not mandatory but polite*/
2  #include <stdio.h> //Preprocessor commands starting with a \#
3  #define PI 4.0*atan(1.0) // Note no semicolons ";", why?
4
5  int main()
6  {
7
8  }
```

Program structure

```
1  /* Description - not mandatory but polite*/
2  #include <stdio.h> //Preprocessor commands starting with a \#
3  #define PI 4.0*atan(1.0) // Note no semicolons ";", why?
4
5  int main() //The main function must be there
6  { // <- the opening bracket
7      // Body of a function, "the meat"
8  } // <- The closing bracket
9
10 // ==== Above obligatory, below additional user defined
    functions ====
11
12 int sum_ints(int a, int b)
13 {
14     return a+b;
15 }
```

Program structure

```
1  /* Description - not mandatory but polite*/
2  #include <stdio.h> //Preprocessor commands starting with a \#
3  #define PI 4.0*atan(1.0) // Note no semicolons ";", why?
4
5  int sum_ints(int a, int b); //Function prototypes, a promise to
    the compiler
6  int a=5; // Definition of global variables if needed. Not
    mandatory, more later.
7
8  int main() //The main function must be there
9  { // <- the opening bracket
10     // Body of a function, "the meat"
11 } // <- The closing bracket
12
13 // ==== Above obligatory, below additional user defined
    functions ====
14
15 int sum_ints(int a, int b)
16 {
17     return a+b;
18 }
19
20 ....
```


Headers

```
1  /* Description - not mandatory but polite*/  
2  #include <stdio.h>  
3  #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 "mygreatestheader.h"` - look for user generated headers in `./`

Functions

```
1
2  int sum_ints(int a, int b); //A prototype end with semicolon
3
4  //type name (arguments)
5  int main(void) // main can have arguments
6  {
7      return 0; // main is special!
8  }
9
10 //this function is of integer type
11 int sum_ints(int a, int b) //it accepts two arguments of integer
    type
12 {
13     return a+b; // since it has a type it must have a return.
14 }
```