Preview of C programming language

Week 01 – Lab 1

Link: https://goo.gl/ZHa3E8

Why C?

- Core of all popular operating systems is written in C

Debian Linux	Android
C : 23,938,129 LOC	**C/C++ : 23,451,409 LOC**
C++: 15,041,861 LOC	Java: 6,891,568 LOC
Python: 1,441,963 LOC	Python: 815,304 LOC
Java: 773,945 LOC	

Sample C program

```c
/*C program to check whether a number entered by user is prime or not using funct
#include <stdio.h>
void prime();
int main(){
     prime();        //No argument is passed to prime().
     return 0;
}
void prime(){
/* There is no return value to calling function main(). Hence, return type of pri
     int num,i,flag=0;
     printf("Enter positive integer enter to check:\n");
     scanf("%d",&num);
     for(i=2;i<=num/2;++i){
          if(num%i==0){
                flag=1;
           }
     }
     if (flag==1)
          printf("%d is not prime",num);
     else
         printf("%d is prime",num);
     }
```

Exercise 1

Step 1: Login to your VM via SSH

Ubuntu, Mac:

$ssh -X [username]@osc-[N].edu.innopolis.ru

Windows: Putty client

Exercise 1 cont.

Step 2: Create your first C program

$mkdir week1 <create directory week1>

$ls <display the content of the current directory>

$cd week1 <change current directory to week1>

$touch ex1.c <create empty file ex1.c>

$gedit ex1.c & <open text editor on ex1.c>

Ctrl - C to cancel

```
#include <stdio.h>

int main()
{
  printf("Hello World!\n");
  return 0;
}
```

Exercise 1 cont.

- Step 3: Compile and run your program

  $gcc ex1.c -o ex1 <-o ex1 - name of executable file produced. Default: a.out>

  $./ex1

Main differences with Java

Criteria	C	Java
type of language	function oriented	object oriented
basic programming unit	function	class
portability of compiled code	no, recompile for each architecture	yes, bytecode is "write once, run anywhere"
security	limited	built-in to language
compilation	gcc hello.c creates machine language code	javac Hello.java creates Java virtual machine language bytecode
execution	a.out loads and executes program	java Hello interprets byte code
allocating memory	malloc	new
de-allocating memory	free	automatic garbage collection

Built-in things C does not have

- Strings (you need to use array of characters to make a string)
- Threads (you need to use a library to achieve multithreading, e.g. #include <pthread.h>)
- Packages
- Classes, objects
- Type safety
- Garbage collection

C pointers

- A pointer is a variable that points to (i.e., contains the address of) a variable or data structure.
- Example:

```
char c1, c2, *p;
c1 = 'c';
p = &c1;
c2 = *p;
```

» c1, c2 - char variables

» p - pointer variable. To declare a pointer use * symbol

» & operation returns the address of a variable

» * operation returns the value stored in the memory cell, pointed by the pointer

Exercise 2

Compile and run the following program:

```
/* Source code to demonstrate, handling of pointers in C program */
#include <stdio.h>
int main(){
   int* pc;
   int c;
   c=22;
   printf("Address of c:%d\n",&c);
   printf("Value of c:%d\n\n",c);
   pc=&c;
   printf("Address of pointer pc:%d\n",pc);
   printf("Content of pointer pc:%d\n\n",*pc);
   c=11;
   printf("Address of pointer pc:%d\n",pc);
   printf("Content of pointer pc:%d\n\n",*pc);
   *pc=2;
   printf("Address of c:%d\n",&c);
   printf("Value of c:%d\n\n",c);
   return 0;
}
```

Exercise 2 cont.

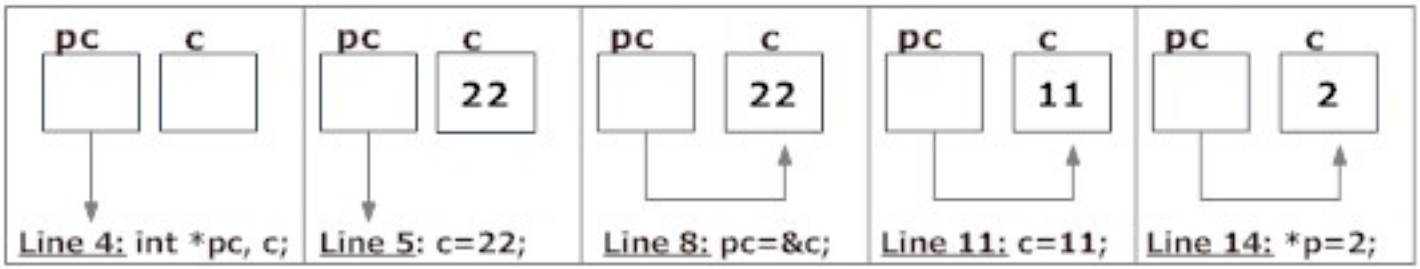
Output

```
Address of c: 2686784
Value of c: 22

Address of pointer pc: 2686784
Content of pointer pc: 22

Address of pointer pc: 2686784
Content of pointer pc: 11

Address of c: 2686784
Value of c: 2
```



Header files

- Typical C project consists of .c and .h files
- .c - source code files
- .h - header files
- Header files contain **declarations**, **definitions** and **macroses** used by multiple source code files

Header files cont.

Main purposes of header files:

- ensure that everyone uses the same version of a data layout definition or procedural code throughout a program.
- easily cross-reference where components are used in a system.
- easily change programs when needed (only one master copy to change).
- save time by not needing to code extensive data layouts (minor, but useful).

Header files example

foo.h

```c
#ifndef FOO_H_   /* Include guard */
#define FOO_H_

int foo(int x);  /* An example function declaration */

#endif // FOO_H_
```

foo.c

```c
#include "foo.h"  /* Include the header (not strictly necessary here) */

int foo(int x)    /* Function definition */
{
    return x + 5;
}
```

main.c

```c
#include <stdio.h>
#include "foo.h"  /* Include the header here, to obtain the function declaration */

int main(void)
{
    int y = foo(3);  /* Use the function here */
    printf("%d\n", y);
    return 0;
}
```

Exercise 3

ex3.h

```
#define FALSE  0
#define TRUE  !FALSE
#define ONE_HUNDRED 100
#define max(A,B)        ( (A) > (B) ? (A):(B))
```

ex3.c

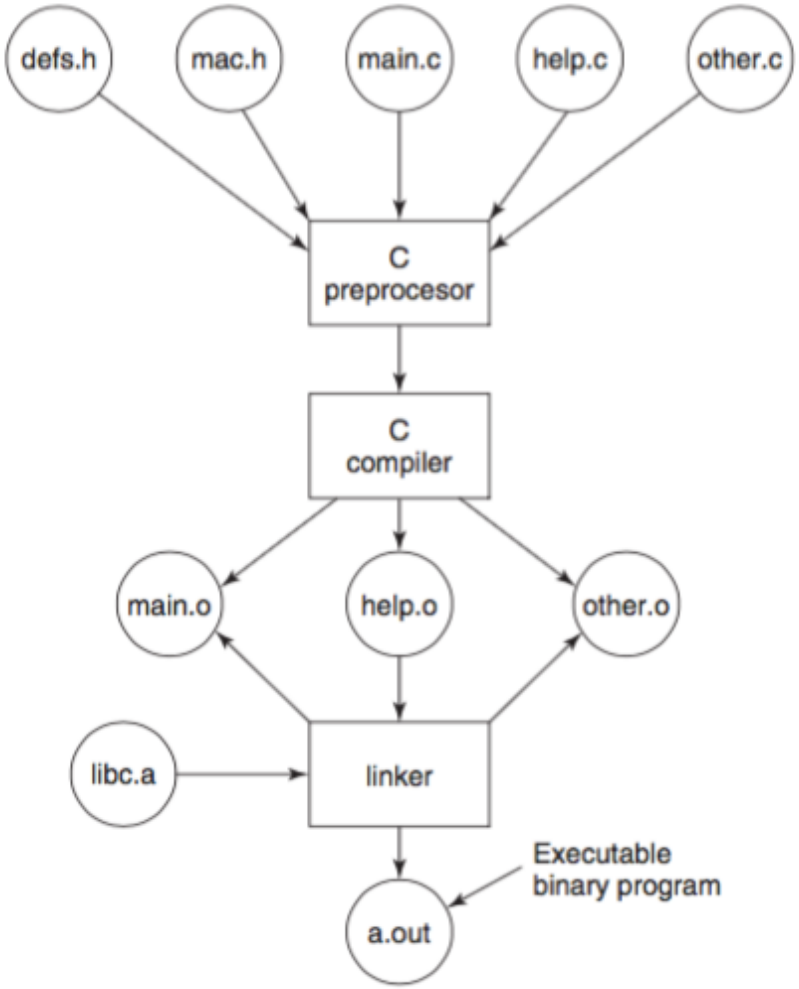
```
#include <stdio.h>
#include "ex3.h"

int main() {
        int a, b, c;
        a = 51; b = 50;
        c = 0;
        printf("%d\n", max(a,b));
        if (c == FALSE) {
                printf("c is false\n");
        }
        if (c == TRUE){
                printf("c is true\n");
        }

        printf("%d\n", ONE_HUNDRED);
        return 0;
}
```

```
$gcc ex3.c -o ex3
$./ex3
```

C compilation process



Exercise 4

- Declare a function *void hello(char* name)* in .h file. Make an implementation of this function that prints “Hello, <name>” to the console in.c file. Create main.c file containing *int main()* function and call *hello(<your name>)* function from *main()*. Compile and run your program.