EECS2031 A Software Tools

SU 2019

Apr 29, 2019 Lecture 1.

1

The Course
EECS2031 Software Tools

- Lecture: CB 121. Mondays 18:00 – 20:00
- Labs: LAS 1006
  - Lab 01 Mondays 15:00 – 17:00
  - Lab 02 Wednesdays 18:00 – 20:00
- Course website: **www.eecs.yorku.ca/course/2031**
  - http://www.eecs.yorku.ca/course_archive/2018-19/S/2031
  - Visit frequently for announcements, due dates, solutions etc …
  - We don't use Moodle/Wiki for this course.

YORK UNIVERSITÉ UNIVERSITY

2

The instructor

- Office: LAS (CSEB) 2015
- Email: **huiwang@eecs.yorku.ca**

- Office hours
  - Mondays before and after class
    - 17:30-18:00 in LAS2015
    - 19:50- ? in CB121
  - By appointment

- COSC/CSE/EECS2031 student, TA, instructor

YORK UNIVERSITÉ UNIVERSITY

3

Course content

- Introduces software tools that are useful in the software development process.

The course covers the following topics:

- ANSI-C
  - Learning how to write C programs
  - *(C Basics, stdio, pointers, memory management, C libraries)*
- Unix (Linux) operating system
  - Using Unix tools to automate compilation, execution and testing
  - *(Commands/utilities, filters and pipes, Shell programming under Unix -- Bourne (again) shell scripts)*
- (Testing and debugging)

YORK UNIVERSITÉ UNIVERSITY

4

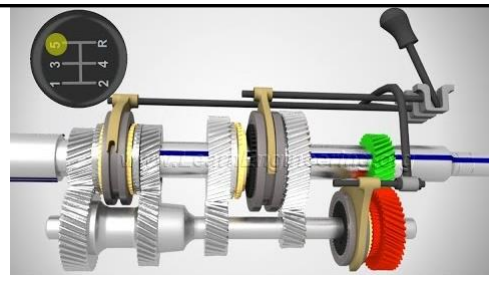
Why 'Software Tools', why now

- EECS 1020/1021 → EECS 1030/1022
  - Basic programming skills / concepts
  - read API specification (client),
  - implement API (implementer).
- Java gives you a "safe" programming environment (VM)
  - higher level

- Now good time to learn how to deal with lower layers (memory, CPU management etc).
  - Some domains require working on lower layer
  - Better understanding of higher layer
  - Lay foundation for future courses, researches, careers, ……

5

Why C and Unix



- Right platforms to teach skills necessary for practical program development
  - C: Expose students to underlying layers/raw machine
    - C's ability to handle low-level activities (direct memory access, memory allocation etc)
    - Safety layers not present (C has poor error detection and significantly fewer safeguards than Java)
      - ✓ A good language to learn testing and debugging
  - Unix: where C runs.
    - Good environment to learn systematic testing

YORK UNIVERSITÉ UNIVERSITY

6

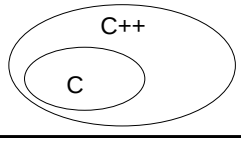
Course objective

- By the end of the course, you should be able to
  - Develop modest-sized programs in C
  - Use UNIX shell commands/utilities
  - Develop programs using UNIX shell scripting language
  - Test and debug C and other programs using UNIX command/scripts

7

What you can gain

- C, Unix for courses/researches/careers/
  - Computer Organization, OS, Embedded system courses
  - Embedded systems, image processing…
  - ....
- Better understanding of programming, including Java
  - `z = a++;  z += 2;  c >> 2;  c= a>b ? c: d`
  - `Student s1 = s2;  s1.id++; s2.id?`
  - `"Pass/call by value" vs. "pass/call by reference"`
- Automatically learn some C++ !!!
  - Lots opportunities



C → C++ → Java

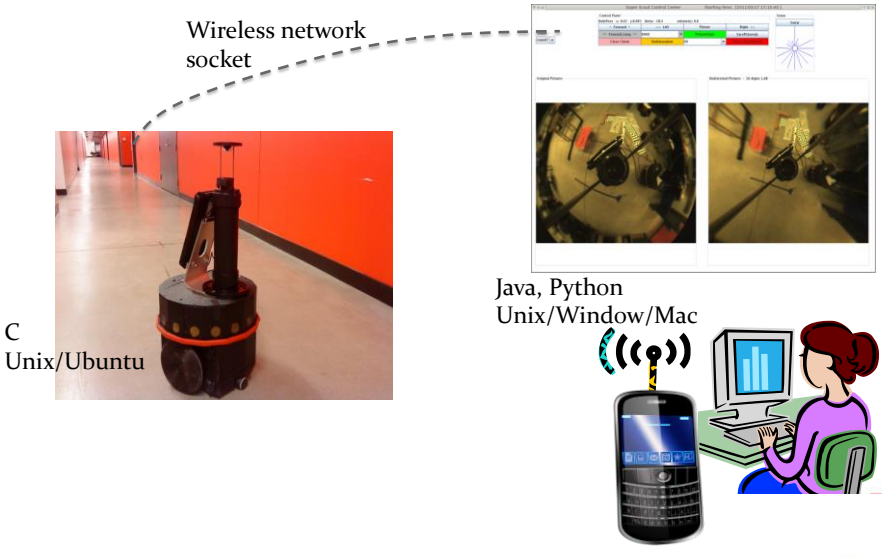
8

Some Applications

- Embedded systems, Network, image processing …..
- An example -- Driving robots
  - Robot side: Unix (Ubuntu), C, shell script, make file ….
  - Base station side: Java, Python, ….
  - Socket programming ….



9



10

rovers

C++, Python
on
Unix/Ubuntu
(ROS)

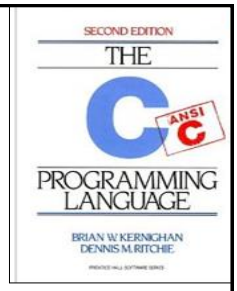
11

drones

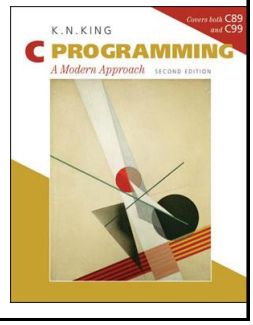
C++, Python
on
Unix/Ubuntu
(ROS)

12

Textbooks



- ***The C Programming Language (2nd Ed)***
  - B. W. Kernighan and D. M. Ritchie (K&R)
  - ISBN 0-13-110362-8
  - Short and well written, covering ANSI-C (C89)
  - Classic C book. Bible
- Alternative C text: *A modern approach (2nd)*
  - Well written. 800 pages
- Unix: another 800 pages book …
- List of recommended books on course web
  - Supplement to the textbooks



13

Administrivia

- Components and weights (Tentative)
  - 12~15% – Weekly labs (8~9)
    - 1~2% each
  - 35~40% – Midterm test
    - 20~25% – Written
    - 10~15% – Lab test I
  - 50~55% – Final exam
    - 35~40% – Written
    - 10~15% – Lab test II
  - ?% attendance/in-class quizzes

YORK UNIVERSITÉ UNIVERSITY

14

Administrivia

- Weekly Labs/exercises/assignments
  - Released 1-2 days after lecture.
  - Each lab is worth about 1~2%.
  - Hands-on learning process for the preceding lecture.
    - Open book. Discussion allowed. Ask questions to TAs or me!
  - Strongly encouraged to come to the lab sessions.
  - Due about a week
  - Check schedule on 'weekly lab' page.

15

Administrivia

- Labtests:
  - Scheduled on lab time TBD
  - Labtest mode: no internet
- Written tests and exams
  - Midterm test in class
    - Right after reading week
  - Final exam Jul 31~ Aug 10
- Let's worry about them later…..

16

Overall, Challenging but doable course

- C bit shifting, (notoriously scary) pointers and memory allocation
  - `int i = k >> 5;`
  - `int * p = &x;  int * p[];`
  - `current -> next = (int *) malloc(sizeof(struct`

- Unix utilities: `grep, sort, wc, chmod, | pipe , `` * ?`
- Unix shell syntax bit strange and strict

```
count=1
while [ $count -lt 100 ]
do
    count = `expr $count + 1`
    echo $count
done
```

YORK UNIVERSITÉ UNIVERSITY

17

Useful suggestions

- Visit website frequently.
  - Probably the only place I announce things to you outside lectures and labs
- Read the lecture notes and the textbook!
  - Following notes. Use textbook for details
  - Notes will be finalized shortly after class

- Come to the class, come to the lab
- Practice, practice, and practice!

18

- Any questions so far?

YORK UNIVERSITÉ UNIVERSITY

19

Overview of C K&R ch1.1-1.8,ch7.1-7.4

- System programming language
  - Originally used to write Unix and Unix tools
  - Later also a popular application programming language
- History of C

BCPL → B → C → K&R C → ANSI C (C89/90) → C99 → C11
1960 1970 1972 1978 1988-89 1999 2011

(NB)

- ANSI-C (C89) standard by *American National Standard Institute*

20

Overview of C K&R ch1.1-1.8,ch7.1-7.4

- Features
  - Low level
    - with high-level constructs and ability to handle low-level activities (direct memory access, memory allocation etc).
  - Small
    - limited set of features
- Strength
  - Efficient and fast
  - Integration with UNIX
- Weakness
  - Permissive, Error-prone &, if (x=1) int i=3.2

21

Overview of C

- Predecessor of modern object oriented languages
  - Many languages derived from C (e.g., C++, Java, Objective-C)
  - C → C++ → Java → C#

- Features of C
  - Something same as or similar to Java (Java adopted)
    - Variable, data type, operator (arithmetic, relational, logical etc), operation precedence, expressions, flow control, …
      - `int, double, int i = 2; i++; i += 2;`
      - `void aFunction (int a)…. int aFunction() ….`
      - `if else, for…, while, do while, case switch,`
  - Something different from Java (not adopted)

YORK UNIVERSITÉ UNIVERSITY

22

Overview of C

- Some differences with Java
  - Procedure-oriented vs. Object Oriented
    - ✓ No classes
    - ✓ no Encapsulation, Inheritance, Polymorphism
  - No garbage collection
  - No exceptions (try - catch)
  - No type **`String`**
  - No type **`Boolean`** – for ANSI C (C89)
  - No //, only /* */ multi line – for ANSI C (C89)
  - Declare all variables at the block beginning -- for ANSI C (C89)
  - Declare or define a function before its first use

  - Has (explicit) pointers
  - Can do (low level) memory allocation and de-allocation
  - Pre-processing, header files, global variables
  - ……

Additional resources on website

YORK UNIVERSITÉ UNIVERSITY

23

Topics of C

- Introduction and Basic I/O - Chapters 1 and 7
- Variables, Types and operators - Chapter 2
- Control flow - Chapter 3
- Functions - Chapter 4
- Arrays and pointers - Chapter 5
- Structures - Chapter 6
- I/O, files - Chapter7
- Dynamic memory allocation (extra)
- Self-referential structures -- Linked list (extra)

YORK UNIVERSITÉ UNIVERSITY

24

C basics

- **The first program – what it looks like**
- Compile and run C program
- Basic syntax
  - Comments
  - Variables
  - Functions
  - Basic IO functions
  - Expression
  - Statements
  - Preprocessing: #include, #define

YORK UNIVERSITÉ UNIVERSITY

25

First C program -- what it looks like?

#include <iostream> in C++

```c
#include <stdio.h>
/*import standard io header files*/

/* salute the world */

int main (int argc, char** argv)
{
    printf( "Hello, world\n" );
    return 0;
}
```

hello.c / first.c
any name .c

```java
import java.lang.….
/*import library functions*/

public class Hello
{

    public static void main(String[] args)
    {
      // System.out.println("Hello World!");
       System.out.printf ("Hello, world\n");
    }
}
```

Hello.java

YORK UNIVERSITÉ UNIVERSITY

26

First C program -- what it looks like?

#include <iostream> in C++

```c
#include <stdio.h>
/*import standard io header files*/

/* salute the world */

int main (int argc, char** argv)
{
    printf( "Hello, world\n" );
    return 0;
}
```

hello.c / first.c
any name .c

```java
import java.lang.....
/*import library functions*/

public class Hello
{

    public static void main(String[] args)
    {
      // System.out.println("Hello World!");
      System.out.printf ("Hello, world\n");
    }
}
```

Hello.java

27

First C program -- what it looks like?

#include <iostream> in C++

```c
#include <stdio.h>
/*import standard io header files*/

/* salute the world */

 main ()
{
    printf( "Hello, world\n" );

}
```

hello.c / first.c
any name .c

```java
import java.lang.....
/*import library functions*/

public class Hello
{

    public static void main(String[] args)
    {
      // System.out.println("Hello World!");
      System.out.printf ("Hello, world\n");
    }
}
```

Hello.java

28

Another C program -- a function

cal.c

```c
#include  <stdio.h>


int sum (int i, int j) {
    int k;
    k = i+j;
    return k;
}

/* main */
main()   {
   int x=2, y=3;
   int s = sum(x,y);
   printf("Sum is %d\n", s);
}
```

Cal.java

```java
import java.lang.….
public class Cal
{
    static int sum(int i, int j) {
        int k;
        k = i + j;
        return k;
     }

     public static void main(String[] args)
        int x=2, y=3;
        int s = sum(x,y);
       // System.out.println("Sum is " + s);
        System.out.printf("Sum is %d\n", s)
      }
}
```

29

C basics

- The first program – what it looks like
- **Compile and run C program**
- Basic syntax
  - Comments
  - Variables
  - Functions
  - Basic IO functions
  - Expression
  - Statements
  - Preprocessing: # include, # define

YORK UNIVERSITÉ UNIVERSITY

30

Compiling and running a C program (general)

- C programs (source code) are in files ending with **.c** --- **hello.c**
- To compile a C program (in our lab):

  % **gcc hello.c**

  - If no syntax error, complier creates an executable program named **a.out** (in the current directory)
- To run

  % **./a.out** or **a.out**

```
indigo 234 % gcc hello.c
indigo 235 % a.out
Hello, world
```

% **gcc hello.c -o hello**

- create an executable program named **hello** (in the current directory)

```
indigo 302 % gcc -o hello hello.c
indigo 303 % hello
Hello, world
```

Either before or after hello.c

31

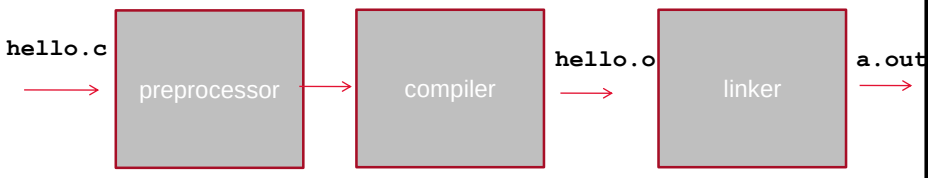
cc hello.c also works in our lab,
cc is a 'symbolic link' to **gcc** --- when you use **cc**, you are using **gcc**

```
indigo 307 % man gcc

NAME
       gcc - GNU project C and C++ compiler

SYNOPSIS
       gcc [-c|-S|-E] [-std=standard]
           [-g] [-pg] [-Olevel]
           [-Wwarn...] [-pedantic]
           [-Idir...] [-Ldir...]
           [-Dmacro[=defn]...] [-Umacro]
           [-foption...] [-mmachine-option...]
           [-o outfile] infile...

       Only the most useful options are listed here; see below for the
       remainder.  g++ accepts mostly the same options as gcc.

DESCRIPTION
       When you invoke GCC, it normally does preprocessing, compilation,
       assembly and linking.
```

32

How C programs are compiled

- C executables are built in three stages



Handles
#include
#define

Converts C code into binary processor instructions -- **hello.o**

- Puts multiple object files together,
- linking object files with necessary system libraries (e.g., code of **printf()**)
- creates an executable program e.g., **a.out**
- remove **hello.o**

We will look at this later

33

C → K&R C → ANSI C (C89/90) → C99 → C11

gcc -- GNU C and C++ compiler, Only C compiler for Linux.

- Support different standards and more
- Default: C89/90 + some C99 features

```
gcc hello.c
```

  - A variable declared as long as before its use.
    Mix variable declarations with uses (C99 feature).
  - // comment (C99 feature)
  - …..

- To compile using ANSI (C89):

```
gcc -ansi  hello.c
gcc -std=c89 hello.c
```

- To compile using C99 :

```
gcc -std=c99 hello.c
```

For your information

YORK UNIVERSITÉ UNIVERSITY

34

Compiling and running. C vs. Java

	C	Java

Program	**hello.c:**
	`#include <stdio.h>`
	`int main(int argc, char**argv){`
	`  printf("Hello, world\n");`
	`  return 0;`
	`}`
		hello.java:
		`public class hello {`
		`  public static void`
		`  main(String[] args) {`
		`  System.out.printf(`
		`  "Hello, world");`
		`  }`
		`}`

Compile	`% gcc hello.c`
	`% ls`
	`hello.c  a.out`
	`%`
		`% javac hello.java`
		`% ls`
		`hello.java  hello.class`
		`%`

Run	`% a.out`
	Hello, world
	`%`
		`% java hello`
		Hello, world
		`%`

35

Compiling and running C

- You need basic knowledge of unix/linux command.
  `cd cd .. cd . ls  cat  rm mkdir cp mv pwd < >` .....
  - If you need to recap, start off with the guided lab tour (CSE1020) posted on the course website. Can ignore file protection (**`chmod`**)

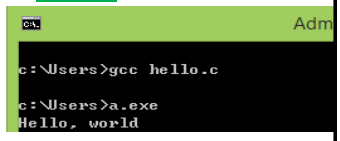
- BTW, do you know
  `grep  wc  sort  cut  find  cmp  uniq |  tr awk sed` .....
  - Don't worry for now.

36

Work off campus?

- Remotely:
  - use PuTTY (and the like) to connect to server (red) and work online
  - Nano, vi, emacs as text-based editor
  - Graphical? Emulator for xterm (e.g., MobaXterm)

- Locally (@home, laptop):
  - (not recommended) use a windows compiler for C codes
    - A good choice could be GCC compiler called MinGW
    - `cmd (dos)` → `gcc` → `a.exe`
  - Ubuntu
    - different flavors
  - For MAC
    - Xcode



Let me know if you need help

37

Work off campus?

- OK, but, all submitted work
  - **must compile in our lab** **default: (C89 + some C99)**

```
gcc hello.c
```

  - are welcomed to follow ANSI-C (C89) but not required

```
gcc -ansi    hello.c
gcc -std=c89 hello.c
```

- Kind suggestion for working mostly on your machine:
  - Wrap up and compile in our lab for final deliverable version
  - Be careful with file transfer: dos file to unix – dos2unix

Let me know if you need help

  - Gets you better prepared for the labtests and unix

38

C basics

- Compile and run C program
- Basic syntax
  - **Comments**
  - Variables
  - Functions
  - Basic IO functions
  - Expression
  - Statements
  - Preprocessing: # include, # define

YORK UNIVERSITÉ UNIVERSITY

39

Comments

- ANSI-C (C89) /* comment */
- Span multiple lines /* …..
  ….. */
- May not be nested ***/* /* */ */***
- Good practice to comment things. But don't write trivial ones

- *//* may not work. Depend on compiler.
  - ANSI-C (C89) ✗
  - C99 ✓
  - In our lab?
    `gcc hello.c` ✓ – default C89 + some C99.
    `gcc -ansi hello.c` ✗

YORK UNIVERSITÉ UNIVERSITY

40

C basics

- Compile and running Comments
- Basic syntax
  - Comments
  - **Variables**
  - Functions
  - Basic IO functions
  - Expression
  - Statements
  - Preprocessing: # include, # define

YORK UNIVERSITÉ UNIVERSITY

41

C variables

- Store data, whose value can change.
  - Declaration and initialization.
    - `int x;  x = 5;`
    - `int x =5; x = 9;`
- Variable names
  - combinations of letters (including underscore character _), and numbers.
  - that do not start with a number; avoid starting with _;
  - are not a keyword.
  - upper and lower case letters are distinct (`x ≠ X`).

Same in Java, C++

- Examples: Identify valid and invalid variable names
  - `abc, aBc, abc5, aA3_ , my_index`
  - `5sda, _360degrees, _temp, char, struct, while`

42

C variables

Basic types in C (what about Java? These four and more...)

- `char` -- **characters**
- `int` -- **integers**
- **floating point**
  - `float` -- **single precision floating point numbers**
  - `double` -- **double precision floating point**

We will further study and discuss other types next week.

43

C variables -- literals

- `int x;  x= 20;` or `int i = 20;`
- `double d = 223.3;`
- `char c = 'b'  c= ' '  c='\n'` (new line) `c= '\t'` (tab).

- Sequence of characters forms strings
  - `printf("hello world\n");`
- But `String s = "hello"` 
  - No string type!!!
  - Array of chars. Will look at it later

One thing to get adapted from Java (among many other things)

44

C basics

- Compile and running Comments
- Basic syntax
  - Comments
  - Variables
  - **Functions**
  - Basic IO functions
  - Expression
  - Statements
  - Preprocessing: #include, #define

45

functions

```
return_type  functionName (parameter type name, ……)
{body block}
```

```
int main(){…}

int sum (int i, int j)
{
  int s = i + j;
  return s;              /*  return i+j;  */
}

void display (int i)
{
  printf("this is %d", i);
}
```

- Similar to Java's (static) methods ? Sort of

46

functions

One thing to get adapted from Java (among many other things)

- ***Must be declared or defined before we can use !*** – *different from Java*
  - ***C89, C99***

- Declaration (prototype) – describe arguments and return type, but no body
  - `int sum (int i, int j);` or `int sum(int, int);`
  - `void display(double i);` or `void display(double);`

- Definition – describe arguments and return value, and gives the code

```
int sum (int i, int j){
   return i+j;          /*  int s = i + j; return s; */
}

void display (double i)
{ printf("this is %f", i);
}
```

- `<stdio.h>` contains declarations (prototypes) for `printf()`, `scanf()` etc. --- that why we "#include" it

47

functions

```
/* Contains declaration
(prototype) of printf() */
```

```
#include <stdio.h>

/* function definition */
int sum (int i, int j){
   return i+j;
}

main()
{
  int x = 2, y = 3;
  int su  = sum(x , y);
  printf("Sum is %d\n", su);
}
```

Defined before (first) function call

OK

(First) function call

48

functions

```
#include <stdio.h>

main()
{
  int x =2, y=3;
  int su = sum(x,y);
  printf("Sum is %d\n", su);
}

/* function definition */
int sum (int i, int j){
   return i+j;
}
```

/* Contains declaration (prototype) of printf() */

Not Defined or Declared before (first) function call

(First) function call

Little luckier if return int…

Defined after (first) function call

49

functions

```
#include <stdio.h>

/* function declaration */
int sum(int, int);     /* int sum(int a, int b) */

main()
{
  int x =2, y=3;
  int su = sum(x,y);
  printf("Sum is %d\n", su);
}

/* function definition */
int sum (int i, int j){
   return i+j;
}
```

/* Contains declaration (prototype) of printf() */

Declared before (first) function call

(First) function call

Defined after (first) function call

50

C basics

- Compile and running Comments
- Basic syntax
  - Comments
  - Variables
  - Functions
  - **Basic IO functions**
  - Expression
  - Statements
  - Preprocessing: # include, # define

YORK UNIVERSITÉ UNIVERSITY

51

Basic I/O functions

<stdio.h>

- Every program has a Standard Input: keyboard
- Every program has a Standard Output: screen
  - Can use redirection Unix < inputFile > outputFile

- **int printf (char *format, arg1, ….);**
  - Formats and prints arguments on standard output (screen or > outputFile)
  - **printf("This is a test %d \n", x)**

- **int scanf (char *format, arg1, ….);**
  - Formatted input from standard input (keyboard or < inputFile)
  - **scanf("%x %d", &x, &y)**

Others (for today)

- **int getchar();**
  - Reads and returns the next char on standard input (keyboard or < inputFile)

- **int putchar(int c)**
  - Writes the character c on standard output (screen or > outputFile)

YORK UNIVERSITÉ UNIVERSITY

52

format string

/* conversion specification */

- **`printf("This is a test %d \n", x)`**
  - Formats and prints arguments on standard output (screen or > outputFile)
  - back in Java 1.5 (year 2005)

https://docs.oracle.com/javase/8/docs/api/java/io/PrintStream.html#printf-java.lang.String-java.lang.Object...-

printf

```
public PrintStream printf(String format,
                          Object... args)
```

A convenience method to write a formatted string to this output stream using the specified format string and arguments.

Parameters:

format - A format string as described in Format string syntax

args - Arguments referenced by the format specifiers in the format string. If there are more arguments than fo
ignored. The number of arguments is variable and may be zero. The maximum number of arguments is limited by th
defined by *The Java™ Virtual Machine Specification*. The behaviour on a null argument depends on the conversion

Returns:

This output stream

Since:

1.5

```
System.out.printf("This is test %d today\n", x)
```

- also introduced in Java 1.5

```
String s = String.format("This is test %d today\n", x);
```

53

format string

/* conversion specification */

- **`printf("This is a test %d \n", x)`**
  - Formats and prints arguments on standard output (screen or > outputFile)
  - Returns number of chars printed (often discarded)

- Format string contains: 1) regular chars 2) conversion specifications
  - **%d next argument is an integer – decimal**
  - **%c next argument is a character**
  - **%f is a floating point number (float, double)**
  - **%s next argument is a "string"**
  - **…**

- `printf("Hello World\n");`
- `printf("%s", "Hello World\n");`

- `printf("Adding %d to %f gets %f\n", 2, 3.5, 5.5);` // replace in order

```
red%  Adding 2 to 3.5 gets 5.5
```

54

functions

/* Contains declaration (prototype) of printf() */

```
#include <stdio.h>

/* function declaration */
int sum(int, int);    /* int sum(int a, int b) */

main()
{
  int x =2, y=3;
  int su = sum(x,y);
  printf("Sum of %d and %d is %d\n", x, y, su);
}

/* function definition */
int sum (int i, int j){
   return i+j;
}
```

55

functions

/* Contains declaration (prototype) of printf() */

```
#include <stdio.h>

/* function declaration */
int sum(int, int);    /* int sum(int a, int b) */

main()
{
  int x =2, y=3;
  //int su = sum(x,y);
  printf("Sum of %d and %d is %d\n", x, y, sum(x,y));
}

/* function definition */
int sum (int i, int j){
   return i+j;
}
```

56

scanf()

- `int x;`
- `scanf("%d", &x)`
  - opposite to `printf()`
  - Wait for standard input (keyboard or < inputFile), then assign input value to `x`
- Format string contains: 1) regular chars 2) conversion specifications
  - `%d` **next argument is an integer – decimal**
  - `%c` **next argument is a character**
  - `%f` **is a floating point number (float. %lf for double)**
  - `%s` **next argument is a "string"**
- `&x` → memory address of `x`.
  - Details later. Take as it is for now.

YORK UNIVERSITÉ UNIVERSITY

57

```
#include <stdio.h>


main()
{
 int a; int b;
 printf("Please enter the number: " );

 scanf( "%d",  &a);    /* assign value to a */

 b = a * 2;
 printf("double of input %d is %d\n", a, b);
 }
```

- `&a` → memory address of `a`. Details later. Take as it is for now.

```
indigo 316 % gcc scanf.c
indigo 317 % a.out
Please enter the number: 34
double of input 34 is 68
indigo 318 %
```

YORK UNIVERSITÉ UNIVERSITY

58

Java version?

```
import java.util.Scanner
public class Scan {

 public static void main(String[] args) {

  Scanner scan = new Scanner(System.in);

  System.out.print("Please enter the number: ");

  int a = scan.nextInt();
  int b = a * 2;

  System.out.printf("double of input %d is %d\n",a, b);
 //System.out.println("double of input "+ a + " is " +  b);

 }   // bufferedReader, console
}
```

YORK UNIVERSITÉ UNIVERSITY

59

Read two ints

```
#include <stdio.h>

int sum (int, int); /* function declaration (prototype) */

main()
{
  int a, b;
  printf("Please enter two integers separated by blank: " );

  scanf( "%d %d",  &a, &b);      /* assign value to a b  */

  printf("Entered %d and %d. Sum is %d\n", a, b, sum(a,b));
}

int sum (int i, int j)
{
  return i+j;
}
```

```
indigo 329 % gcc scanf2.c
indigo 330 % a.out
Please enter two integers separated by blank: 4 32
Entered 4 and 32. Sum is 36
indigo 331 %
```

60

Read two ints

```
#include <stdio.h>

int sum (int, int); /* function declaration (prototype) */

main()
{
  int a, b;
  printf("Please enter two integers separated by --: " );

  scanf( "%d--%d",  &a, &b);      /* assign value to a b  */

  printf("Entered %d and %d. Sum is %d\n", a, b, sum(a,b));
}

int sum (int i, int j)
{
  return i+j;
}
```

```
indigo 332 % gcc scanf3.c
indigo 333 % a.out
Please enter two integers separated by --: 4--32
Entered 4 and 32. Sum is 36
indigo 334 %
```

61

getchar, putchar (Ch 1.5)

- **`int getchar(void)`**
  - To read one character at a time from the *standard input*
  - Returns the next input char each time it is called;
  - Returns EOF when it encounters end of file.
    - End of input file; or keyboard: Ctrl-d (Unix) or Ctrl-z (Windows).
    - EOF: an int defined in <stdio.h>, value is -1.
- **`int putchar(int c)`**
  - Puts the character c on the *standard output*
  - Returns the character written (usually ignored);
  - **`printf("%c", c);`**

YORK UNIVERSITÉ UNIVERSITY

62

getchar, putchar (Ch 1.5)

- **countChar.c**

```
#include <stdio.h> // define EOF

main(){
 int c;
 int count = 0;

 c = getchar();
 while(c != EOF) /* no end of file*/
 {
   count++; //include spaces and '\n'
   c = getchar(); /* read next */
 }
 printf("# of chars: %d\n",count);
}
```

red 309 % **a.out**
hello ↵
how are you ↵
i am good ↵
^D
red 310 %
# of chars: 28

Redirected from file

red 312 % **cat greeting.txt**
hello ↵
how are you ↵
i am good ↵
red 313%

redirect input from a file

red 314 % **a.out < greeting.txt**
# of chars: 28

red 315 % **a.out < greeting.txt > out.txt**

redirect output to a file

red 316 % **cat out.txt**
# of chars: 28

63

getchar + putchar (Ch 1.5)

- **copy.c**

```
#include <stdio.h>

main(){
 int c;
 c = getchar();
 while(c != EOF)
 {
   putchar(c);
   c = getchar();/*read next*/
 }
}
```

red 309 % **a.out**
hello ↵
hello
how are you ↵
how are you
^D
red 310 %

Actually get/put chars line by line!

Redirected from file

red 313 % **cat greeting.txt**
hello
how are you
i am good

redirect input from a file

red 314 % **a.out < greeting.txt**
hello
how are you
i am good

red 315 % **a.out < greeting.txt > out.txt**

red 316 % **cat out.txt**
hello
how are you
i am good

redirect output to a file

64

Summary and ``future work``

- Today: course introduction. C basics
  - Variables
  - Functions: declaration vs. definition
  - Basic IO functions
    - **`scanf & printf,`**
    - **`getchar & putchar`**

- Next lecture (next week):
  - Finishes C basic syntax
  - C data, type, operators (Ch 2)
  - C control flow (Ch 3)

YORK UNIVERSITÉ UNIVERSITY

65

- Get textbook K&R and read Ch1 (1.1-1.5,1.7. getchar/putchar, functions). Ch7 (7.1, 7.2, 7.4. standard IO, printf, scanf).
  You can also read Ch3 (control flow).

- Read guided lab tour for CSE1020.

- Try the code in Ch1 and notes about basic IO (printf(), scanf(), getchar(), putchar())!

- First Weekly Lab/assignment to be released tomorrow or Wednesday, due in about a week.
  - TAs in lab this Wednesday and next Monday (May1,6) to help out. You are encouraged to come.

YORK UNIVERSITÉ UNIVERSITY

66