CS 24000 - Programming In C

Week One: Introduction

Zhiyuan Li
Department of Computer Science
Purdue University, USA

PURDUE
UNIVERSITY

Acknowledgement

- *This introductory lecture has many slides (especially those on course policies) borrowed from professors who taught CS240 before me.*
- *Professors Jan Vitek and Janannathan are acknowledged in particular.*
- *Our lead TA Ahmed Hussein has provided background info on the admin aspects of this course.*

- *We will have 6 graduate-student TAs and we'll work hard to make this a good learning experience.*

PURDUE
UNIVERSITY

Lecture Outline

- Course objectives and outline
- Course policies and workload
- Introduction to the programming and execution environment for C programs
- Introductory C program examples

PURDUE
UNIVERSITY

Course objectives and outline

- Department's course description for CS240

  The UNIX environment, C development cycle, data representation, operators, program structure, recursion, macros, C preprocessor, pointers and addresses, dynamic memory allocation, structures, unions, typedef, bit-fields, pointer/structure applications, UNIX file abstraction, file access, low-level I/O, concurrency.

- Department's course objectives for CS240

  Students understand programming principles and techniques for problem solving in the C programming language.

- Prerequisites

  Undergraduate level CS 18000 Minimum Grade of D- or
  Undergraduate level CS 18200 Minimum Grade of D- [may be taken concurrently]

PURDUE
UNIVERSITY

How we administer CS240

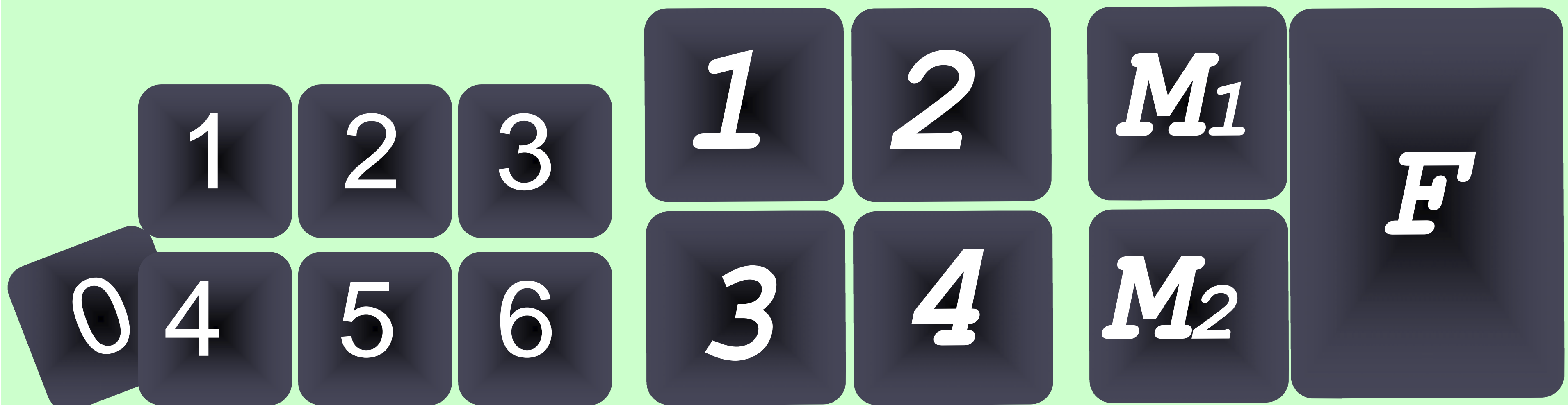
- From the prerequisites, we can see that the entry point to CS240 is not very high. Therefore we expect a wide range of "CS maturity" among participants
- We will have to target the main body of the students
  - Some may feel the pace to be a little too slow
  - Some may feel the pace to be a bit too fast
- We will use a forum called "Piazza" to interact with students from various background
- The labs will be the main place to have in-person help from the TAs
  - Go to lab hours even if you are not "officially" enrolled in those hours.

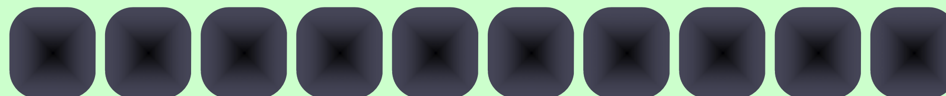
PURDUE
UNIVERSITY

Workload

6

- 6 labs, 4 projects, 2 midterms, 1 final, 10 reading quizzes
- Grade mix:
  - labs 20%, projs 30%, midterms 20%, final 20%, quizzes 10%
  - pick 5 best labs and 8 best quizzes





PURDUE
UNIVERSITY

Late policy

- No late submission is allowed
    - *Rationale: In a large class it is not possible to accommodate requests for extensions*

- All assignments are handed on time
  - You are allowed to miss one lab for any reason
  - You are allowed to miss two quizzes for any reason
    - *Rationale: you can discard your worse grade or miss a class without need for justification*

Academic integrity

- *Any case of cheating will handled by the Dean of students*
- You are encouraged to discuss problems and approaches but:
  - Sharing solution is not allowed.
  - Buying solutions is not allowed.
  - Copying code from the internet is not allowed.
  - Copying code from other students is not allowed.
  - Copying partial code from other students is not allowed.
- `http://homes.cerias.purdue.edu/~spaf/cpolicy.html`
- *First offense get a "-100" for the work. Second offense gets an "F" for the course and a record in Dean of Students Office*

PURDUE UNIVERSITY

Attendance

- Class attendance and Lab attendance is mandatory
  - If you miss class, get someone else's notes…
  - If you miss a lab, try to get in on another session during the same week

- *Rationale:*
  - *Slides are not complete and may contain inaccuracy*
    - *To prepare for the exam, trust your notes and the book*
  - *Announcements are made in class.*
    - *The website may be out of date*
    - *Piazza may not have all of the information*

PURDUE UNIVERSITY

Resources

- Lecture notes and useful links are placed on my home page:
  - www.cs.purdue.edu/homes/ci
- Grade books are placed on Blackboard Learn
  - Go to *www.purdue.edu* and then find the "Blackboard" button
- To find all lab hours for CS240 and yours in particular
  - Go to *www.purdue.edu* and then find the "MyPurdue" button
- For help
  - Lab sessions are your first line of defense
    - Ask as many questions as you can… come prepared
  - When desperate or need to appeal a grade
    - send email to *admin240s13@cs.purdue.edu* to make an appointment
      - We'll see if there is a need for regular TA office hours outside the lab
  - But, use Piazza as the default place for Q&A

PURDUE UNIVERSITY

Piazza

- Rather than emailing questions to the teaching staff, I encourage you to post your questions on Piazza. If you have any problems or feedback for the developers, email team@piazza.com.

- Find our class page at:
  - https://piazza.com/purdue/spring2013/cs240/home
- Enroll yourself
- Ask questions either in public or private

PURDUE
UNIVERSITY

Why Piazza?

- allows both students and TAs to ask and answer questions at all times and greater
- flexibility
- students get faster answers, TAs get better management of their time
- prevents students from just bringing laptops with half-written code showing up and
  - saying “Fix this”
- allows questions to be answered by fellow students and endorsed by TAs
- allows answers to be public and reduces repeated questions
- prevents any one student from consuming all the available TA hours
- allows posted material to be announced via email, AND also publicly posted (more next page)

PURDUE
UNIVERSITY

Why Piazza (Cont'd)

- preventing "I didn't get that email"
- gives a simple area for lab FAQs
- prevents the TA with the best English skills from being swamped
- encourages students to ask well formed question about their code

PURDUE
UNIVERSITY

Autograder

- We'll use "Autograder" to grade assignments and projects
- Pros:
  - encourages students to keep working on an assignment until they get it right
  - gives students hints into code failures
  - gives students experience with failures from unseen data
  - lets students experiment with code designs
  - greatly simplifies TA workload in grading
  - reduces bias in grading
  - eliminates most arguments for late turn-ins
- Cons:
  - limited viewing of code style by TAs
  - some limits on the nature of assignments (must be autogradable)
  - Feedbacks may be tricky

PURDUE
UNIVERSITY

Labs

- Lab sessions will involve short problems that can usually be done in the lab session (but sometimes take longer)

- 20% of grade

  - *Rationale: ensures you can get help from TAs while solving the problem*

Projects

- Projects are 2 week long programming exercises that will culminate in a small group assignment
- Projects are done outside of the lab sessions and should take around 12 hours each
- 30% of grade

Midterms

- Midterms are in class, closed-book, exams that are representative of the difficulty of the final
- 20% of grade

Final

- A closed-book exam
- 20% of grade

Reading quizzes

- Short in-class tests of your understanding of the reading assignment for the day using clickers
- Quizzes cover material that has **not** been presented in class
- It is your responsibility to read the book and ask questions ahead of class in case something is unclear
- Quizzes will take place in the first five minutes of class and are usually unannounced
- *Don't come late*

10% of grade

Questions

- Use the following algorithm
  - Ask on Piazza
  - Ask TA at lab
  - Ask Prof during class
    - *Rationale: This focuses the interaction and ensure best use of your time*
- Regrading questions
  - Regrading will only be done **the week following** release of the grade
  - Send mail to the course staff mail alias (*admin240s13@cs.purdue.edu*) with a precise description of your grading request, and get a ticket
  - Midterm/Final issues are dealt by the Instructor, project/labs are dealt by your TA and can be escalated to the Faculty member
    - *It is your responsibility to check your grades!*

PURDUE
UNIVERSITY

Questions

- How to ask a question on Piazza:
  - Read the book, slides, notes
  - Describe the problem clearly, using the right terms
  - Add code in attached files
  - Add output from compiler
  - Add any other relevant information

- Be polite and respectful of TAs' time and we'll do the same
- Avoid anonymous questions...
- Avoid #IDW questions...

Environment

- Linux
- gcc
- gdb
- emacs/vi/jedit
- Firefox
- VMware image

The use of a VM development environment:

- allows students to develop/test code in the same environment that it is graded in
- allows students to test code in both 64-bit (lab) and 32-bit (VM) environments
- give all students the same toolset
- prevents “But it worked on my OS!” excuses
- is platform independent (Win/Mac/Linux)
- encourages learning the UNIX environment

PURDUE
UNIVERSITY

Why Teach C?

- You have already studied programming in Java, why C now?
- There are two main differences between Java and C
  - Java is executed mainly by *interpretation*
    - C (and its extension C++) is executed as a compiled (and optimized) sequence of instructions
  - Java is highly abstract, with system/hardware details hidden
    - C can interact with the system/hardware much more directly
- Hence, C is more suitable as a system implementation language
  - OS, device drivers, etc
  - Java virtual machine
  - Embedded systems (think of the future innovations of smart devices)
  - Mission critical and high performance applications

PURDUE
UNIVERSITY

Using C Programming as the Medium

- We can better understand
  - How a program interacts with system/hardware
  - How a program is converted from a high-level format to a low-level format
  - What is the run time mechanism as the program executes, e.g.
    - The memory allocation for static, local, and dynamic variables
    - The memory layout for various data structures
    - The calling/returning sequences for function/method invocations
  - How a program processes data of various types
- Simply speaking, a program’s functionality is to take data from outside world (incl. control signals), process it, and generate new data (incl. control signals) for the outside world

PURDUE
UNIVERSITY

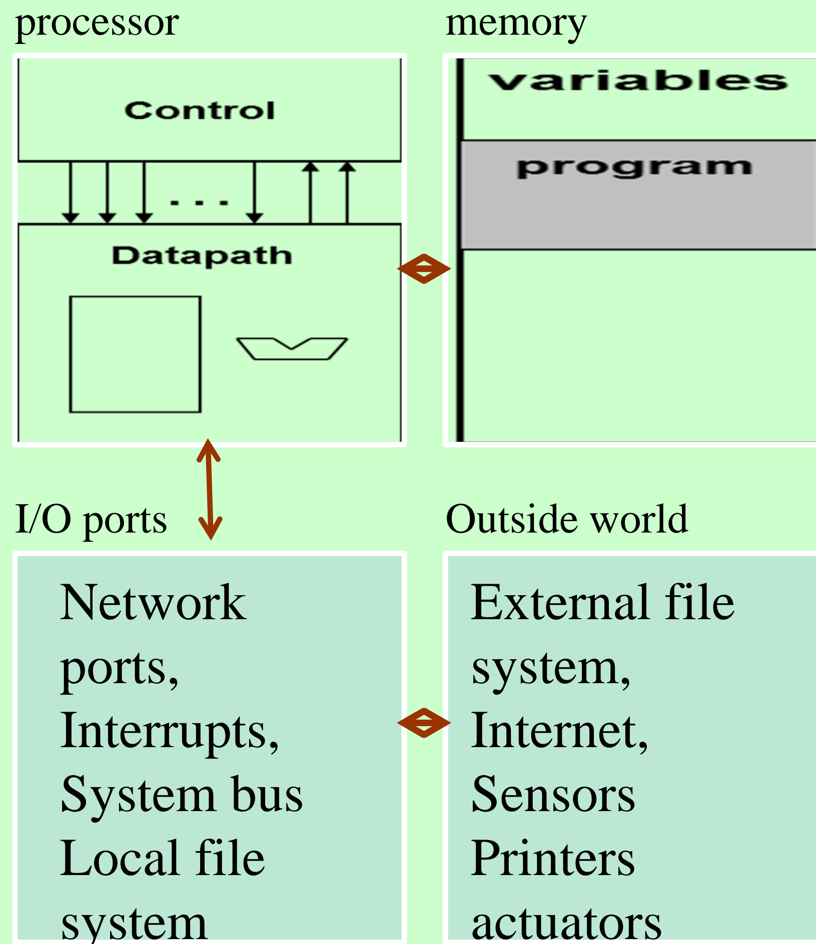
- As the program processes data, intermediate results are stored internally in the form of what we call *variables*.
- Variables must reside in the memory hierarchy to be accessed by the processing hardware
- Conceptually, variables are mapped to the *virtual address space,* which is a linear sequence of bytes.
- *Now let us look at the diagram next page and try to visualize how data move in and out on the computer and how the internal data, i.e. variables and program code move inside the computer*
  - Let us imagine various kind of programs, including system programs and applications.
  - Let us think of directly executed programs versus interpreted programs

PURDUE
UNIVERSITY

About I Clicker

- Please register your I clicker on Blackboard Learn (go to our class)
- Please bring your clicker next Tuesday to try out in class (so we can enroll you on my iclicker program)

Computer Architecture



PURDUE UNIVERSITY

Instruction Set as the Interface Between Hardware and Software

Algorithms

Programs at high level

User program at machine level

OS at machine level

ISA

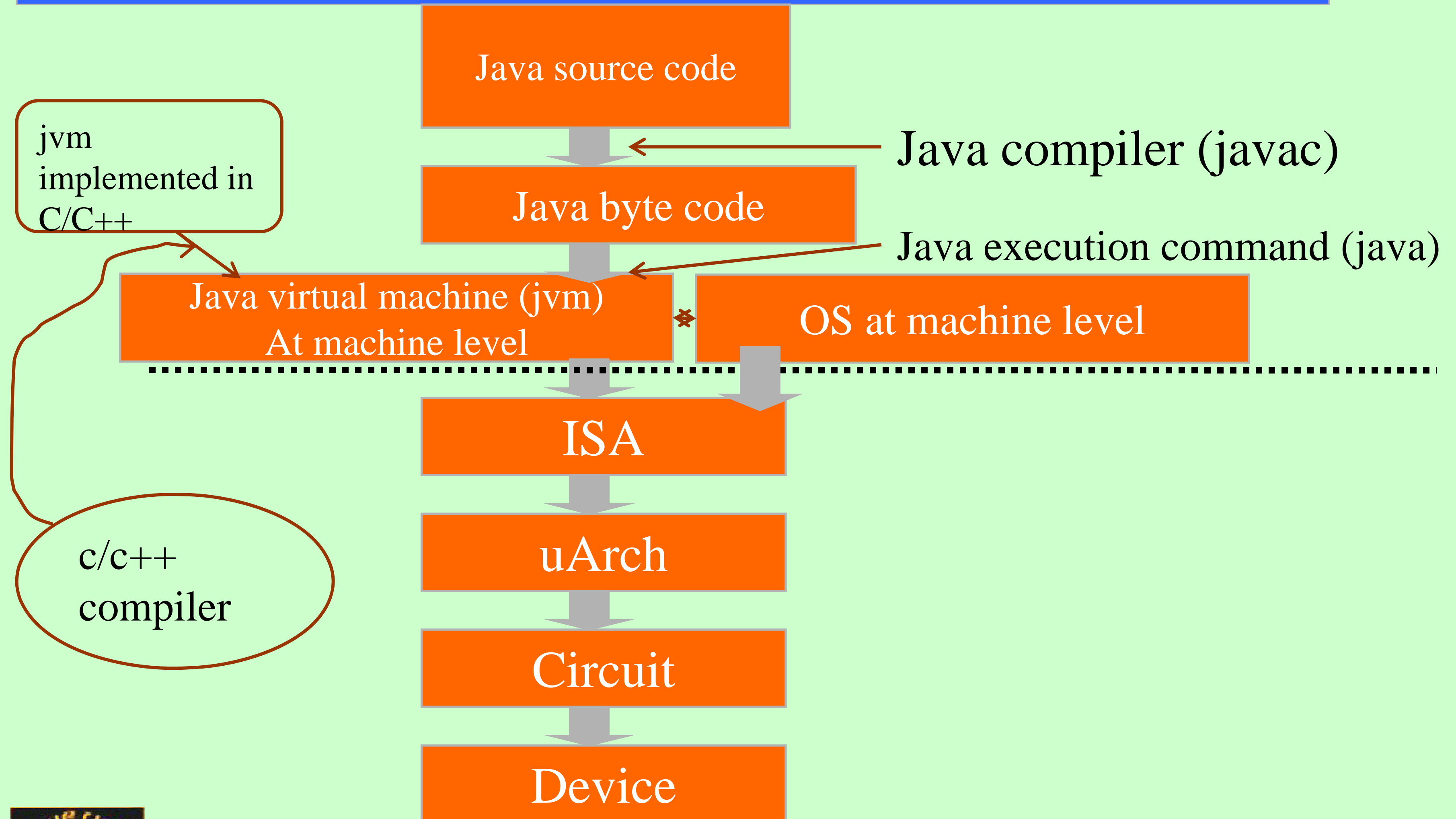
uArch

Circuit

Device

PURDUE UNIVERSITY

Java Execution Model



PURDUE
UNIVERSITY

- Without knowing the internal mechanism of how a program interacts with the system/hardware, we will lose
  - Many career opportunities
  - Many innovation opportunities
    - Think of future computing and communicating devices
  - Ability to diagnose tricky software errors
  - Ability to prevent security hazards

Difference between assembly code and C code (An example: the Fibonacci function)

```
 n←r1
 t1←n<3
 if t1 goto L1
 r1←t1-1
 call f
 t2←r1
 r1←t1-2
 call f
 goto L2
L1:  r1←1
L2: return
```

```
Int f(int n) {
  if (n < 3) return (1);
  else return (f(n-1)+f(n-2));
}
```

PURDUE
UNIVERSITY

- We see that C still hides a lot of low level details of machine operation.
- Therefore, although C is highly flexible and efficient for
  - manipulating data at low level details and
  - interacting directly with the system, but, at the same time,
- it is still a high-level language suitable for building large software

Start with a very simple example

- Let us start with a very simple example from Chapter 8 to see how in C we can directly interact with Unix OS
- *“In the UNIX operating system, all input and output is done by reading or writing files, because all peripheral devices, even keyboard and screen, are files in the file system. This means that a single homogeneous interface handles all communication between a program and peripheral devices.”*

- “Since input and output involving keyboard and screen is so common, special arrangements exist to make this convenient.”
- “When the command interpreter (the ``shell'') runs a program, three files are open, with file descriptors 0, 1, and 2, called the standard input, the standard output, and the standard error. If a program reads 0 and writes 1 and 2, it can do input and output without worrying about opening files.”
- “The user of a program can redirect I/O to and from files with < and >:

prog <infile >outfile

PURDUE
UNIVERSITY

- “In this case, the shell changes the default assignments for the file descriptors 0 and 1 to the named files. Normally file descriptor 2 remains attached to the screen, so error messages can go there.”
- “ The file assignments are changed by the shell, not by the program. The program does not know where its input comes from nor where its output goes, so long as it uses file 0 for input and 1 and 2 for output.”

PURDUE
UNIVERSITY

8.2 Low Level I/O - Read and Write

- "Input and output uses the read and write system calls, which are accessed from C programs through two functions called read and write."
- " For both, the first argument is a file descriptor. The second argument is a character array in your program where the data is to go to or to come from. The third argument is the number is the number of bytes to be transferred."

```
int n_read = read(int fd, char *buf, int n);
int n_written = write(int fd, char *buf, int n);
```

PURDUE UNIVERSITY

- int n_read = read(int fd, char *buf, int n);
- “Each call returns a count of the number of bytes transferred. On reading, the number of bytes returned may be less than the number requested. A return value of zero bytes implies end of file, and -1 indicates an error of some sort.”
- For writing, the return value is the number of bytes written; an error has occurred if this isn't equal to the number requested.
- int n_written = write(int fd, char *buf, int n);

- “Any number of bytes can be read or written in one call. The most common values are 1, which means one character at a time (``unbuffered"), and a number like 1024 or 4096 that corresponds to a physical block size on a peripheral device.”
- “Larger sizes will be more efficient because fewer system calls will be made.”
- Using these two syscalls, we can write a simple program to copy its input to its output, the equivalent of the Unix command *“cat”*. (I modified the code to fit my system.)
- “This program will copy anything to anything, since the input and output can be redirected to any file or device.”

PURDUE
UNIVERSITY

```
/* #include "syscalls.h" I don't need
this on my machine*/
#define BUFSIZ 100
main() /* copy input to output */
{
char buf[BUFSIZ];
int n;
while ((n = read(0, buf, BUFSIZ)) > 0)
   write(1, buf, n);
return 0;
}
```

PURDUE
UNIVERSITY

- Let us try compile and run the program
  - First by typing text on the keyboard
  - Next by copying a JPEG image to another
- Let us use the command “size” to see the size of the machine code (in binary) as it is loaded to the memory for execution
  - We see it is only 1823 bytes
  - The smallest jvm would be larger than this by a factor of a few dozens
- The size is different between the “size” showing and the “ls” command showing, because the a.out file has other things to assist loading

PURDUE UNIVERSITY

- From this simple program we see several features in C
  - Comments
  - Macros
  - The main fuction
  - Declaration statements and executable statements
- We briefly explain how the main function gets invoked, but leave the details to the OS course
- Next let’s try to “view” some non-text file by “vi” or “more”.
  - We’ll see garbled output
  - What is a text file?
    - A sequence of ASCII characters, each taking a byte.

CHARACTER REPRESENTATION ASCII

ASCII (American Standard Code for Information Interchange)

MSB (3 bits)

LSB (4 bits)

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	‘	P
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	I	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	m	n	~
F	SI	US	/	?	O	n	o	DEL

char

- The basic unit of data on the UNIX system is a byte (or *char* in a C program)
  - *Does not have to be an ASCII char*
- Therefore it is useful to have a high-level C function, called *getchar*, to read one character at a time from the standard input

```
/* getchar: unbuffered single character input */
int getchar(void)
{
char c;
return (read(0, &c, 1) == 1) ? (unsigned char) c : EOF;
```

PURDUE
UNIVERSITY

- In this short function, we see
  - A new data type called “void”
  - A strange expression that has “?”
  - A typecast operation
  - A new data type called “unsigned char”
  - A symbolic constant EOF
- Some of these things will be explained later in the course
  - For historical reasons (mainly to help the compiler when compiler techniques are still not mature), the C language has quite some “features” that are deemed by many as error prone.
  - One of the things we must do in this course is to point out the potential pitfalls
  - It is a good idea to avoid using cryptic syntax in a C program

Be very careful and remember that getchar returns an integer!

PURDUE
UNIVERSITY

Standard C Functions

- C comes with a set of prewritten standard functions that can be "included" in the program we write
- E.g. a much more sophisticated version of *getchar* is a part of the "standard I/O functions"

#include <stdio>

- The "include" macro tells the C compiler's *pre-processor* to include (i.e. to *inline*) the set of I/O standard functions in the program text before compiling
- *printf* is yet another highly useful standard I/O function in <stdio>
- We now go back to the beginning of the book and trying out more program examples and explain them.

PURDUE
UNIVERSITY

Using printf

```
#include <stdio.h>
main()
{
printf("hello, world\n");
```

- *printf* is actually a quite complex function. It is a “formatted output function”, to make our task of printing various types of data easy.
- *printf* is a special case of *fprintf*
- *int fprintf(FILE *stream, const char *format, ...)*

- It is notable that *fprintf* has a variable length of parameters
  - After the format parameter, there may be 0 or more variables as parameters
- Let us go to Appendix B.1 to learn something about <stdio.h>
  - "A stream is a source or destination of data that may be associated with a disk or other peripheral. The library supports text streams and binary streams, although on some systems, notably UNIX, these are identical."
  - "A text stream is a sequence of lines; each line has zero or more characters and is terminated by '\n'. An environment may need to convert a text stream to or from some other representation (such as mapping '\n' to carriage return and linefeed)."

PURDUE
UNIVERSITY

- “A stream is connected to a file or device by opening it; the connection is broken by closing the stream. Opening a file returns a pointer to an object of type FILE, which records whatever information is necessary to control the stream. We will use ``file pointer'' and ``stream'' interchangeably when there is no ambiguity.”
- “When a program begins execution, the three streams stdin, stdout, and stderr are already open.”
- Formatted output:

  int fprintf(FILE *stream, const char *format, ...)

  int printf(const char *format, ...)

  printf(...) is equivalent to fprintf(stdout, ...).

PURDUE UNIVERSITY

- char *format means the (parameter) variable "format" is a pointer to a sequence (of unknown length) of characters
- const *format means the parameter is a quoted string
- So, printf expects the first argument to be a quote string which may include textual characters, escape sequences (e.g. \n), and *conversion characters.*
  - *Each conversion character defines the formatting specification for the corresponding parameter that follows the "format" parameter*
  - Our current example contains no conversion characters.
  - String is perhaps the most complicated (and difficult to understand) data type in C, due to its variable length that is often not predetermined.
  - Please follow Section 1.2 and try out variants of our example yourself

PURDUE
UNIVERSITY

Example of using format specification

```
#include <stdio.h>
/* print Fahrenheit-Celsius table
for fahr = 0, 20, ..., 300 */
main()
{
   int fahr, celsius;
   int lower, upper, step;
   lower = 0; /* lower limit of temperature scale */
   upper = 300; /* upper limit */
   step = 20; /* step size */
   fahr = lower;
   while (fahr <= upper) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr, celsius);
       fahr = fahr + step;
   }
}
```

PURDUE
UNIVERSITY

- We will not explain in length syntax that is similar to Java
- We see a new escape sequence \t   for printing a tab
- There are two conversion characters used in the quoted string corresponding to fahr and celsius respectively
  - %d   (same as %i)   is for int; signed decimal notation.
- For other conversion characters, see Table B.1

PURDUE
UNIVERSITY

A floating point number example

```
#include <stdio.h>
/* print Fahrenheit-Celsius table
for fahr = 0, 20, ..., 300; floating-point version */
main()
{
   float fahr, celsius;
   float lower, upper, step;
   lower = 0; /* lower limit of temperature scale */
   upper = 300; /* upper limit */
   step = 20; /* step size */
   fahr = lower;
   while (fahr <= upper) {
       celsius = (5.0/9.0) * (fahr-32.0);
       printf("%3.0f %6.1f\n", fahr, celsius);
      fahr = fahr + step;
   }
}
```

- In this example, we see
  - Implicit type conversion (from int to float)
  - The floating point format specification
  - Width modifier for the conversion characters
    - You can find the description of the width modifier in the book
- There are five different sizes of integer types (signed and unsigned):
  - char, short, int, long, and long long
  - Internally, char is a single byte, others depending on whether the machine is a 32-bit or a 64-bit machine, e.g. on a 32-bit machine, int is normally 4 bytes, long could also be 4 bytes, short is 2 bytes, long long could be 8 bytes

A For statement example

```
#include <stdio.h>
/* print Fahrenheit-Celsius table */
main()
{
int fahr;
for (fahr = 0; fahr <= 300; fahr = fahr + 20)
    printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

PURDUE
UNIVERSITY

- In this example, we see
  - the For loop header that has three components
    - Initial value of an induction variable, fahr,
    - The termination condition, fahr <= 300
    - The increment operation for the induction variable
    - Any component could be left as blank
  - and the loop body, which is a C statement
    - It could be a compound statement with { } to enclose a list of C statements
  - We also see an expression used as one of the arguments in printf

PURDUE
UNIVERSITY

Character I/O

```
#include <stdio.h>
/* copy input to output; 1st version
*/
main()
{
    int c;
    c = getchar();
    while (c != EOF) {
        putchar(c);
        c = getchar();
    }
}
```

- int getchar(void)
- int putchar(int c)

- Please try out the examples and exercises in the textbook up to, and including, section 1.5.
- Lab 0 will be a warm up lab (not graded) to try out the submission and autograder
- Lab 0 will be the base for the graded Lab 1 in the week that follows Lab 0.
  - Therefore, it is a good idea to work on Lab 0 seriously.

PURDUE
UNIVERSITY