Introduction to C

CSE 2031
Fall 2012

1

C and Unix History

- Widely used, powerful, and fast.
- Both started at AT&T Bell Labs.
- UNIX was written in assembly, later changed to C.
- Many variants of UNIX.

2

C vs. Java

- Java-like (actually Java has a C-like syntax), some differences
- No //, only /* */ multi-line and no nesting
- No garbage collection
- No classes
- No exceptions (try … catch)
- No String type

3

First C Program

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

Note: **#include <filename.h>** replaces the line by the actual file before compilation starts.

4

1

Special Characters

\n	New line
\t	Tab
\"	Double quote
\\	The \ character
\0	The null character
\'	Single quote

5

Basic Input and Output

CSE 2031
Fall 2012

6

Basic I/O

- Every program has a *standard input* and *output.*
- Usually, keyboard and monitor, respectively.
- Can use > and < for redirection

- getchar()
- putchar()
- printf()
- scanf()

7

getchar( ) (7.1)

- To read one character at a time from the *standard input* (the keyboard by default):

  int getchar(void)
- returns the next input char each time it is called;
- returns EOF when it encounters end of file.
  - EOF input: Ctrl-D (Unix) or Ctrl-Z (Windows).
  - EOF value defined in <stdio.h> is -1.

8

2

putchar(c) (7.1)

- Puts the character c on the *standard output* (the screen by default).

  int putchar(int)
- returns the character written;
- returns EOF if an error occurs.

9

Example

```
#include <stdio.h>
#include <ctype.h>
main() /* Convert input to lower case */
{
 int c;
 c = getchar();
 while ( c != EOF ) {
  putchar( tolower(c) );
 c = getchar();
 }
}
```

10

Example: more compact code

```
#include <stdio.h>
#include <ctype.h>

main() /* Convert input to lower case */
{
 int c;
 while ((c = getchar()) != EOF)
  putchar(tolower(c));
}
```

11

Behaviour of getchar()

- getchar() buffers input characters until a new line or EOF is entered.
  - That is, nothing happens until you hit Return or EOF.

- If the buffer is currently empty and EOF is entered, the program terminates.

- If the buffer is not empty, it is processed, but execution continues.

I/O Redirection

prog < infile
- prog reads characters from infile instead of the standard input.

prog > outfile
- prog writes to outfile instead of the standard output.

otherprog | prog
- Output from otherprog is the input to prog.

prog | anotherprog
- puts the standard output of prog into the standard input of anotherprog.

13

printf() and scanf()

printf("This is a test %d \n", x)
scanf("%x %d", &x, &y)

%d   %s   %c   %f   %lf
integer string character float   double precision
%x
hexadecimal integer

printf( ) (7.2)

int printf(char *format, arg1, arg2, ...);

- Converts, formats, and prints its arguments on the standard output under control of the **format**.

- Returns the number of characters printed (usually we are not interested in the returned value).

- Very similar to Java **printf**

15

printf Conversions (Pg. 154)

Character	Argument type; Printed As
d,i	int; decimal number
o	int; unsigned octal number (without a leading zero)
x,X	int; unsigned hexadecimal number (without a leading 0x or 0X, using abcdef or ABCDEF for 10, ...,15.
u	int; unsigned decimal number
c	int; single character
s	char *; print characters from the string until a '\0' or the number of characters given by the precision.
f	double; [-]m.dddddd, where the number of d's is given by the precision (default 6).
e,E	double; [-]m.ddddddE+/-xx or [-]m.ddddddE+/-xx, where the number of d's is given by the precision (default 6).
g,G	double; use %e or %E if the exponent is less than -4 or greater than or equal to the precision; otherwise use %f. Trailing zeros and a trailing decimal point are not printed.
p	void *; pointer (implementation-dependent representation).
%	no argument is converted; print a %

16

4

Output Formatting with printf( )

- A **minus sign**, which specifies **left adjustment** of the converted argument.
- A number that specifies the **minimum field width**. The converted argument will be printed in **a field at least this wide**. If necessary it will be padded on the left (or right, if left adjustment is called for) to make up the field width.
- A **period**, which **separates the field width from the precision**.
- A number, the **precision**, that specifies the **maximum number of characters to be printed from a string**, or the **number of digits after the decimal point** of a floating-point value, or the **minimum number of digits for an integer**.

17

scanf( ) (7.4)

- scanf( ) is the input analog of printf( ).

- To read an integer:
```
int num;
scanf("%d", &num);
```
- `&num` is a pointer to `num`.

- To read a char and a float:
```
char c; float f;
scanf("%c %f", &c, &f);
```

18

scanf Conversions (Pg. 158)

Character	Input Data; Argument type
d	decimal integer; int *
i	integer; int *. The integer may be in octal (leading 0) or hexadecimal (leading 0x or 0X).
o	octal integer (with or without leading zero); int *
u	unsigned decimal integer; unsigned int *
x	hexadecimal integer (with or without leading 0x or 0X); int *
c	characters; char *. The next input characters (default 1) are placed at the indicated spot. The normal skip-over white space is suppressed; to read the next non-white space character, use %1s
s	character string (not quoted); char *, pointing to an array of characters long enough for the string and a terminating '\0' that will be added.
e,f,g	floating-point number with optional sign, optional decimal point and optional exponent; float *
%	literal %; no assignment is made.

19

scanf( )

```
int scanf(char *format, arg1, arg2, ...);
```

- reads characters from the standard input, interprets them according to the specification in `format`, and stores the results through the remaining arguments.
- stops when it exhausts its format string, or when some input fails to match the control specification.
- returns the number of successfully matched and assigned input items (e.g., to decide how many items were found).
- returns 0 if the next input character does not match the first specification in the format string (i.e., an error).
- On the end of file, EOF is returned.
- **Note: `arg1, arg2, ...` must be pointers!**

20

5

More examples

- You will look at many more examples in the lab
- A couple more now…

Next time ...

- Types, Operators and Expressions (Chapter 2)
- Testing and Debugging