

### **Portable Programming**

CS 217

#### Language

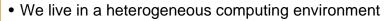


- Stick to the standard
  - Program in high-level language and within the language standard
  - Standard may be incomplete
    - char type in C and C++ may be signed or unsigned

#### • Program in the mainstream

- $\circ\,$  Mainstream implies the established style and the use
  - Program enough to know what compilers commonly do
  - Difficult for large language such as C++
- Beware of language trouble spots
  - Some features are intentionally undefined to give compiler implementers flexibility

## Portability



- $\circ\,$  Multiple kinds of HW: IA32, IA64, PowerPC, Sparc, MIPS, Arms,  $\ldots\,$
- $\,\circ\,$  Multiple kinds of systems: Windows, Linux, MAC, SUN, IBM,  $\ldots\,$
- Software will be used in multiple countries
- It is difficult to design and implement a software system
  - It takes a lot effort to support multiple hardware and multiple operating systems (multiple versions)
  - Patches and releases are frequent operations
- If a program is portable, it requires no change to run on another machine
  - Correctness portability (primary concern)
  - Performance portability (secondary concern)
- Normally, portability is difficult to achieve
  - But, making the programs more portable is a good practice

### Size of Data Types



- What are the sizes of char, short, int, long, float and double in C and C++?
  - They are not defined, except
    - char must have at least 8 bits, short and int at least 16 bits
    - sizeof(char) ≤ sizeof(short) ≤ sizeof(int) ≤
      - sizeof(long)
    - sizeof(float)  $\leq$  sizeof(double)
- In Java, sizes are defined
  - byte: 8 bits
  - char: 16 bits
  - short: 16 bits
  - int: 32 bits
  - long: 64 bits

### **Order of Evaluation**

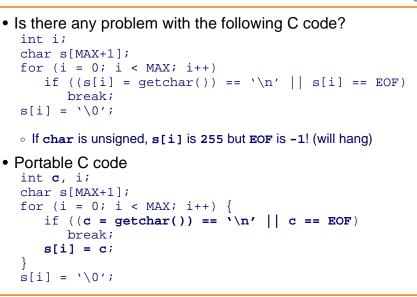


- What does the following code do? n = (getchar() >> 4) | getchar();
  - The order is not specified

```
strings[i] = names[++i];
```

- i can be incremented before or after indexing strings!
- printf("%c %c\n", getchar(), getchar());
  - The second character in stdin can be printed first!
- What are the rules in C and C++?
  - $\circ~$  All side effects and function calls must be completed at ";"
  - $\circ~$  && and || operators execute left to right and only as far as necessary
- What about Java?
  - $\circ\,$  Require expressions including side effects be evaluated left to right
  - $\circ\,$  But, Java manual advises not writing code depending on the order
- Our Advice: do not depend on the order of evaluation in an expression

# Signed or Unsigned?



## Other C Language Issues



- Arithmetic or logical shift
  - $\circ\,$  Signed quantities with >> may be arithmetic or logical in C
  - $\,\circ\,$  Java reserves >> for arithmetic right shift and >>> for logical
- Byte order
  - $\circ~$  Byte order within <code>short</code>, <code>int</code> and <code>long</code> is not defined
- Alignment of items within structures, classes and unions
  - $\circ\,$  The items are laid out in the order of declaration
  - $\circ\,$  The alignment is undefined and there might be holes

```
struct foo {
```

char x;

```
int y; /* can be 2, 4, or 8 bytes from x */
```

## **Use Standard Libraries**



- Pre-ANSI C may have calls not supported in ANSI C
  - $\,\circ\,$  Program will break if you continue use them
  - $\,\circ\,$  Header files can pollute the name space
- Consider the signals defined
  - ANSI C defines 6 signals
  - POSIX defines 19 signals
  - Most UNIX defines 32 or more
- Take a look at /usr/include/\*.h to see the conditional definitions

#### **Use Common Features**



#### Motivation

 Write a program that runs on Unix and on a cell phone and cell phone environment may have fewer libraries and different type sizes

- $\circ~$  Use the common ones
- Avoid conditional compilation
  - $\circ~ \texttt{\#ifdef}$  are difficult to manage because it can be all over the places

some common code #ifdef MAC

... #else

#ifdef WINDOWSXP

... #endif #endif

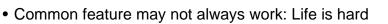
# Data Exchange



9

- Use ASCII text
  - Binary is often not portable
- Still need to be careful
  - But, even with text, not all systems are the same
    - Windows systems use use '\r' or '\n' to terminate a line
    - UNIX uses only '\n'
  - Example:
    - Use Microsoft Word and Emacs to edit files
    - CVS assume all lines have been changed and will merge incorrectly
  - Use standard interfaces which will deal CRLF (carriage-return and line feed) and newline in a consistent manner

## Isolation



- Localize system dependencies in separate files
  - $\,\circ\,$  Use a separate file to wrap the interface calls for each system
  - $\,\circ\,$  Example: unix.c, windows.c, mac.c,  $\ldots\,$
- · Hide system dependencies behind interfaces
  - Abstraction can serve as the boundary between portable and nonportable components
  - Java goes one big step further: use virtual machine which abstracts the entire machine
    - Independent of operating systems
    - Independent of hardware

## **Byte Order**



10

- Recall big-endian and little-endian?
- Consider the following program between two processes
  - Writing a short to stdout: unsigned short x; x = 0x1000; ... fwrite(&x, sizeof(x), 1, stdout)
  - Later, read it from stdin unsigned short x;
    - fread(&x, sizeof(x), 1, stdin);
- What is the value of x after reading?



#### **Byte Order Solutions** More on Byte Order Conditional compilation Language solution · Conditional compilation for different byte orders Java has a serializable interface that defines how data items are Swap the byte order if it is necessary packed • What is the pros and cons of this approach? • C and C++ require programmers to deal with the byte order - Save some instructions · Binary files vs. text files - Make the code messy Binary mode for text files • Fix the byte order for data exchange No problem on UNIX • Sender: - Windows will terminate reading once it sees Ctrl-Z as input unsigned short x; /\* high-order byte \*/ putchar(x >> 8);putchar(x & 0xFF); /\* low-order byte \*/ • Receiver: unsigned short x; x |= getchar() & 0xFF; /\* read low-order byte \*/

#### Internationalization

13

- Don't assume ASCII
  - Many countries do not use English
  - Asian languages use 16 bits per character
- Standardizations
  - Latin-1 arguments ASCII by using all 8 bits (superset of ASCII)
  - Unicode uses 16 bits per character and try to use Latin-1 encoding
  - $\,\circ\,$  Java uses unicode as its native character set for strings
- Issues with unicode
  - Byte order issue!
  - Solution is to use UTF-8 as an intermediate representation or defined the byte order for each character

# Summary

#### • Language

- Don't assume char signed or unsigned
- Always use sizeof to compute the size of types
- $\,\circ\,$  Don't depend on the order of evaluation of an expression
- Beware of right shifting a signed value
- $\circ\,$  Make sure that the data type is big enough
- Use standard interfaces
  - $\circ~$  Use the common features
  - Isolation
- Byte order
  - Fix byte order for data exchange
- Internationalization
  - Don't assume ASCII and English



14

