Portable Programming

CS 217

Portability

- We live in a heterogeneous computing environment
  - Multiple kinds of HW: IA32, IA64, PowerPC, Sparc, MIPS, Arms, ...
  - Multiple kinds of systems: Windows, Linux, MAC, SUN, IBM, ...
  - 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

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
  - 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

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) ≤ 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?
  
  ```c
  n = (getchar() >> 4) | getchar();
  ```
  
  – The order is not specified
  
  ```c
  strings[i] = names[++i];
  ```
  
  – i can be incremented before or after indexing strings!

  ```c
  printf("%c %c\n", getchar(), getchar());
  ```
  
  – The second character in stdin can be printed first!

• What are the rules in C and C++?
  
  ○ All side effects and function calls must be completed at “;”
  ○ && and || operators execute left to right and only as far as necessary

• What about Java?
  
  ○ Require expressions including side effects be evaluated left to right
  ○ 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?

• Is there any problem with the following C code?
  
  ```c
  int i;
  char s[MAX+1];
  for (i = 0; i < MAX; i++)
      if ((s[i] = getchar()) == '\n' || s[i] == EOF)
          break;
  s[i] = '\0';
  ```

  ○ If char is unsigned, s[i] is 255 but EOF is -1! (will hang)

• Portable C code
  
  ```c
  int c, i;
  char s[MAX+1];
  for (i = 0; i < MAX; i++) {
      if ((c = getchar()) == '\n' || c == EOF)
          break;
      s[i] = c;
  }
  s[i] = '\0';
  ```

Other C Language Issues

• Arithmetic or logical shift
  
  ○ Signed quantities with >> may be arithmetic or logical in C
  ○ Java reserves >> for arithmetic right shift and >>> for logical

• Byte order
  
  ○ Byte order within short, int and long is not defined

• Alignment of items within structures, classes and unions
  
  ○ The items are laid out in the order of declaration
  ○ The alignment is undefined and there might be holes

  ```c
  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
  
  ○ Program will break if you continue use them
  ○ 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
  - Use the common ones

- Avoid conditional compilation
  - #ifdef are difficult to manage because it can be all over the places
    - some common code
    - #ifdef MAC
    - ...
    - #ifdef WINDOWS
    - ...
    - #endif
    - #endif

Data Exchange

- 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

- Common feature may not always work: Life is hard

- Localize system dependencies in separate files
  - Use a separate file to wrap the interface calls for each system
  - Example: unix.c, windows.c, mac.c, ...

- Hide system dependencies behind interfaces
  - Abstraction can serve as the boundary between portable and non-portable components
  - Java goes one big step further: use virtual machine which abstracts the entire machine
    - Independent of operating systems
    - Independent of hardware

Byte Order

- Recall big-endian and little-endian?

- Consider the following program between two processes
  - Writing a short to stdout:
    ```c
    unsigned short x;
    x = 0x1000;
    ...
    fwrite(&x, sizeof(x), 1, stdout)
    ```
  - Later, read it from stdin
    ```c
    unsigned short x;
    ...
    fread(&x, sizeof(x), 1, stdin);
    ```

- What is the value of x after reading?
Byte Order Solutions

- Conditional compilation
  - Conditional compilation for different byte orders
  - Swap the byte order if it is necessary
  - What is the pros and cons of this approach?
    - Save some instructions
    - Make the code messy

- Fix the byte order for data exchange
  - Sender:
    unsigned short x;
    putchar(x >> 8);  // high-order byte */
    putchar(x & 0xFF); // low-order byte */
  - Receiver:
    unsigned short x;
    x = getchar() << 8; // read high-order byte */
    x |= getchar() & 0xFF; // read low-order byte */

More on Byte Order

- Language solution
  - Java has a serializable interface that defines how data items are packed
  - C and C++ require programmers to deal with the byte order

- Binary files vs. text files
  - Binary mode for text files
    - No problem on UNIX
    - Windows will terminate reading once it sees Ctrl-Z as input

Internationalization

- 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
  - 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
  - Don’t depend on the order of evaluation of an expression
  - Beware of right shifting a signed value
  - Make sure that the data type is big enough

- Use standard interfaces
  - Use the common features
  - Isolation

- Byte order
  - Fix byte order for data exchange

- Internationalization
  - Don’t assume ASCII and English