Portability

Ravishankar Rajagopalan

When we think about it?

- Programs/software successful at short notice
  - Healthy assumption to make that your program would be run on multiple platforms
- Enterprises tired of maintaining different code bases
  - Combinations of
    - OS
      - Desktop & mobile
    - Platforms - hardware
    - Higher level constructs/interface components
      - E.g.: Browsers, Libraries..
  - Performance gain is not significant to warrant the investment
- Designing for portability
  - Often results in better software designs with less maintenance
- High ROI

Guidelines

- Clean separation of platform specific code and common code
- Avoid adding platform specific code unless absolutely needed
- Use standard interfaces and libraries known to work cleanly
  - Sometimes they may not be the latest API, but stick to stability
- Widely used languages based on open standards have different implementations
- Union and intersection approach
- Exchange data in text rather than in binary

Implementation woes

- *“Different browsers have differing levels of pickiness with regards to how you end arrays and objects. For example, Firefox is more than okay with an array looking like this: [item0, item1,]". However, this same code will make Opera barf because it hates the trailing comma. IE will make the array a three-item array, with the third item undefined! This is bad code for sure, but there's been dynamically generated javascript I've worked on that was a huge pain to rewrite - would've been nice if this just worked.”*

Language quirks

- Data types
- Endian-ness
- Order of evaluation undefined
  - `word[i] = sentence[++i]`
  - `fprintf(w, “%c %c”, getchardata(), getchardata())`
- Shift
  - Arithmetic or logical
- Size of structures and alignments

```
struct S {
     char a;//Filler here -
     int x;//Could be on a 2/4/8 byte boundary
}
```

Story of #ifdef

- Conditional compilation can be tricky
  - To read esp. when interspersed with the programs own conditional statements

```
#ifndef SHAREDSYS
     for (i = 0; i < total_count; i++)
#endif
#ifdef SHAREDSYS
     if items[i] == total {
     //do something
     //
#endif
          {
               if (i == LASTITEM_TAG)
#ifndef SHAREDSYS
                    break;
#endif
          }
```

Story of #ifdef

- Conditional compilation can be risky
  - Worse is #ifdef's get verified only on target environment when enabled

```
#ifdef MAC
    printf("Mac specific execution");
    if (i > MAC_ID) ...
    //Some more code here
#endif
#ifdef WIN
    printf("Windows code");
    if (i > WIN_ID) ...
    //Some more code here

#endif
#ifdef LINUX
    printf("It's a Linux environment");
    if (i > LINUX_ID) ...
    //Some more code here
#endif
```

Better practice

- System dependencies in separate file
  - Typically used for OS, graphic environments/UI components being different
- Move dependencies to interfaces
  - Many languages do it, e.g. their IO and networking libraries, typically run without any change on most systems
- Java is a good example of portable code

Identifying the problem

- Remember this code ?
  - EOF is -1 in stdio and as getchar returns -1 s[i] (unsigned char) stores it as 255

```
int i;
char s[MAX];

for(i = 0; i < MAX - 1; i++)
    if((s[i] = getchar()) == '\n' || s[i] == EOF)
      break;
s[i] = '\0';
```

Fixing the problem finally!

- Moral - careful about language & compiler quirks!

```
int ch, i; //int type is signed in C (need to prefix unsigned to suggest otherwise)
char str[MAXLEN]

for(i = 0; i < MAXLEN - 1; i++) {
    if((ch = getchar()) == '\n' || ch == EOF)
      break;
    str[i] = ch
}
str[i] = '\0';
```

Exchanging data

- Exchanging text preferably
- Exchanging bytes
  - Careful about byte ordering -
    - Can use a fixed byte order for transmission, transmit one character at a time, receiver can assemble it in a well known format
    -
-

Some things about portability

- Different output of multiple existing versions possibly on different platforms
- Interpreting data written by different versions
  - Change of data formats - avoid as it’s a cost on portability
  - Need a strong check on backward compatibility
  - Provide ways to convert if needed
- Different compilers can produce different results
- Careful with internationalization
  - Assuming all data’s always ASCII and English is a mistake

Summary

- Designing for Portability is a time saviour in the long-term
- Needs knowledge of implementation & portability issues in target environment
- Union and intersection approach