How to approach the homework assignments

Review of HW1

- Input: text

```
As I was going down the stair,
   I saw a man who wasn't there.

He wasn't there again today;
   he must be from the CIA.
```

- Output: semi-justified text

```
As I was going down the stair, I saw
a man who wasn't there.

He wasn't there again today; he must
be from the CIA.
```

Questions to answer

- What output width should we use?
- What should we do about
  - multiple spaces between words?
  - spaces at the start of a line?
  - paragraphs with multiple blank lines between them?
- What if a single word is too large?

Defining the problem

- You cannot write a program to solve this problem without answering the questions
  - If you try, your program will answer them one way or another
- It will be easier if you answer them explicitly before you start

Choosing your abstractions

- Once you have answered the questions, your answers will be in terms of concepts such as *line*, *word*, *character*, *space*, etc.
- You can use those concepts as the basis for the abstractions in your program.

Mechanics

- Any language, any computer
- Assignments are usually due on Monday
- Late assignments will not be accepted unless there is a *very* good reason
- Hand in assignments on paper, preferably stapled
- Please save the assignments after you get them back

What to hand in

- Source code (with comments)
- Input, if any
- Output

What we're looking for

- A program that does what was asked…
- …and does it clearly…
- …and with test data that proves it.

What makes a good program?

- It solves the problem correctly
- It is easy to understand
- It is easy to modify

Solving the problem

- It's not fair to solve the problem by hand and insert that solution into the program
- For example, assume the assignment is "write a program that adds 2 and 2"
  - Right: `printf("%d\n", 2+2);`
  - Wrong: `printf("%d\n", 4);`

Ease of understanding

- It's a judgment call
- Some comments are essential
- Too many comments are harmful
- It is particularly useful to explain the purpose of variables and functions

Comments

- Strategic comments are useful

```
// Here we handle the case where…
// The important variables are…
// Elements [0,n) are full
```

- Tactical comments are less so

```
++i;            // increment i
char x[N];      // a character array
```

Modifiability

- Classroom exercises are always artificial
- Still, you can often pretend that you're writing a "real program"

What are "real programs?"

- The people who want them don't always know what they want (even if they're the authors)
- What they want changes over time
- Successful solutions suggest new problems
- Successful programs usually got that way a little at a time

General implications

- Solutions to problems are rarely final
- When writing a program, it is important to think about how it might change
- Well written programs will take plausible future changes into account
- Each aspect that might change should appear in as few places as possible

Homework implications

- Homework assignments are unrealistically small, when compared with commercial projects
- Therefore, you should be more aggressive about imagining future changes than you might be otherwise
- Homework programs should be better organized than their size suggests

An example

- Imagine an assignment to compute the prime numbers < 10000 and print them in columns
- What changes might we imagine for future versions?

Alternative versions

- Compute something other than primes
- Compute more primes (too many to fit in memory)
- A different output format
- Do something with the primes other than print them

Solution implications

- No magic numbers!
- The computation and printing parts of the program should be separate
- There should be a clean interface between the parts
- The program should not keep all the prime numbers in memory at one time; doing so might use lots of memory

What are magic numbers?

- Programs often have *parameters* that characterize them
  - The beginning and end of the range to search for primes
  - The length and width of a page
- Each parameter should appear only once in the program

Localizing parameters

- Right:

```
static const int N = 100;
char buffer[N];
for (int i = 0; i < N; ++i)
    buffer[i] = ' ';
```

- Wrong:

```
char buffer[100];
for (int i = 0; i < 100; ++i)
    buffer[i] = ' ';
```

Parameters in header files

- You can put the parameters into a separate file to avoid copying

```
// file param.h:
static const int N = 100;
// file main.c
#include "param.h"
char buffer[N];
// …
```

Modularity

- If you intertwine computation and printing, it becomes harder to change either one
- It is better to keep them separate and define a clean (i.e. as simple as possible) interface between them

Modularity example

- Right:

```
if (p is prime)
    print(p);
```

- Wrong:

```
if (p is prime) {
    buffer[n] = p;
    if (++n == buffer_size)
        flush_buffer();
}
```

Efficiency

- Efficiency comes in three forms
  - Time
  - Space
  - Effort (in writing the program)
- All three are important

When does it matter?

- Efficiency is usually irrelevant for small inputs
- Therefore, a useful strategy is to decide what parameters might be large, and design a program that works efficiently even in those cases

Example tradeoff

- Storing all the primes has the disadvantage of requiring memory proportional to the number of primes
- Storing none of them slows down the algorithm
- The best tradeoff might be to store the primes up to `sqrt(n)`

Low-level efficiency

- Many compilers will generate faster code for

```
char* p = x;
while (p < x+N)
  *p++ = ' ';
```

than they will for

```
for (int i = 0; i < N; ++i)
  x[i] = ' ';
```

- In practice, it rarely matters

High-level efficiency

- Many people will sort an array this way

```
for (i = 0; i < N; ++i)
  for (j = 0; j < i; ++j)
    if (x[i] < x[j]) {
      int t = x[i];
      x[i] = x[j];
      x[j] = t;
    }
```

- Don't do it!

What's the difference?

- Code generation for accessing array elements might make the program twice as slow
- Using an order n^2 sorting algorithm instead of an order n log n algorithm might make the program millions of times slower

So how should I sort?

- C has a library routine for sorting:

```
void qsort(void *, unsigned, unsigned,
  int (*)(const void *, const void *));
```

- C++ has one too, though you may have to look for it on your implementation:

```
template<class T> void sort(T, T);
```

Sorting integers (C version)

- Write a comparison function

```
int cmp(const void *px, const void *py)
{
    int x = *(int *) px;
    int y = *(int *) py;
    if (x < y)
        return -1;
    if (x > y)
        return 1;
    return 0;
}
```

Sorting integers (part 2)

- Now you can call `qsort` to do the work

```
qsort(x, n, sizeof(int), cmp);
```

- Note that your comparison function must be written to take arguments of type `const void *`, which means that you must convert the arguments yourself inside the function

Sorting integers in C++

- The arguments to `sort` are pointers to the first and one past the last elements of the array

```
sort(x, x+N);
```

- This `sort` function will be part of the standard C++ library; you can find implementations for most current compilers if you look around

Sorting other types in C++

- The `sort` library function is a *template*, which allows the C++ compiler to generate appropriate code during compilation for your particular type
- It can sort arrays of any type, as long as the type has < defined appropriately
- It can sort other sequential data structures too, as we shall see

Floating-point arithmetic

- Be careful about mixing integer and floating-point arithmetic. How *not* to test for primeness:

```
int d = 2;
while (d <= sqrt(n) && n % d != 0)
    ++d;
if (d > sqrt(n)) { /* n is prime */ }
```

- This might fail, for example, if `sqrt(25)` were to yield `4.9999999`

Summary

- A good solution to a problem will
  - solve the problem
  - be easy to change to solve related problems
  - be neither too abstract nor too concrete
- Homework solutions should be somewhat more general than their size suggests