Warning: These notes are not complete, it is a Skelton that will be modified/add-to in the class. If you want to us them for studying, either attend the class or get the completed notes from someone who did

CSE2031

Basic Testing

PPU Ch II.6

These slides are based on slies by Prof. Wolfgang Stuerzlinger

Testing

- Testing is getting sure your code is correct (no bugs).
- In reality, you can only detect the existence of bugs, not their absence.
- Run your code many times using different inputs.

Testing

- Best way to write bug-free code, generate it by **correct (bug-free)** program.
- Do not wait till you complete the program to test it, test every piece that you write (function, block, if, …)
- If you wait until something breaks, you probably have forgotten what the code does.

Testing

- What do you need for testing?
  - The code you want to test
  - Some inputs
  - What is the “correct” output of the above inputs, so you can compare.
- Test Coverage: did you cover every statement in the code?

Random Testing

- Random inputs to the program
- Easy to do
- Without a statistical framework, the results are meaningless.

Black-Box Testing

- Assume no knowledge of the implementation (code)
- Prepare the test based on the specifications.
- Better to do it before the implementation.
- Better if prepared by some one else other than the person who will write (wrote) the code.
- May not test every path in the program.

Glass-Box Testing

- Assume full knowledge of the program.
- Chose test cases to test all different paths in the program.

```
If( a> b) {
    x=….;
   if( c>=d) {
      x=…;
      y=…;
   }
   else {
    ….
   }
else {
…..
}
```

Regression Testing

- When you fix a bug, you may introduce another bug.
- When you fix a bug, you may break another fix
- When you create a test, keep it
- When you fix a bug, apply all previous tests

Boundary Condition Testing

- Reads characters until it finds a new line or fills a buffer.

```
int i;
char s[MAX];
for(i=0; s[i] = (getchar()) != '\n' && i<MAX-1; ++i)
        ;
s[--i]='\0';
```

Boundary Condition Testing

- After fixing it

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

Boundary Condition Testing

- Fixing the EOF

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

Boundary Condition Testing

- What about the case where the input is very large number of characters without a new line.
- Thinking about that might lead to a gap in the specification, must be resolved as early as possible.

Pre- and Post-Conditions

```
double avg(double[] a, int n)   {
    int i;
    double sum;
    sum=0.0;
    for(i=0; i<n; i++)
            sum+=a[i];
    return sum/n;
}
```

What if n=0?

return n<=0 ? 0.0 : sum/n;

The use of assertions

- You can use assertion facilities in <assert.h>
- Use it only when the failure is really unexpected and there is no way to recover
- assert (n>0);
- If that is not true, the program terminates with a message saying the assertion failed.
- Useful for validating properties of interfac.

Example

```
int factorial (int n)
{
   fac =1;
   while(n--)
         fac*=n;
   return fac;
}
```

Example

```
/* print the characters of a string one per line */
 i=0;
  do {
    putchar(s[i++]);
    putchar('\n');
  } while (s[i] != '\0');
```

Example

- **Binary search, what to test for**
- Array with 0 elements
- One element, key is <,=,> the element
- Two elements, try all five combinations
- Duplicate elements, try all different combinations of key
- Three elements, all different combinations
- Four elements, …
- If passed, probably OK