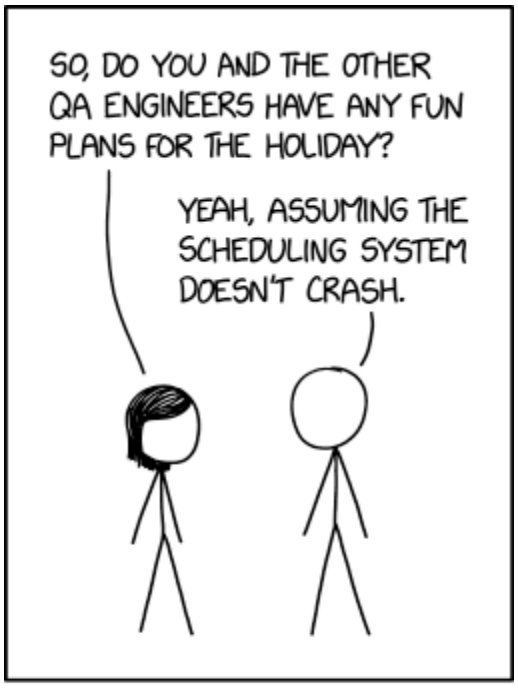

Positive test cases expect good things to happen.

- Nominal function: does it do what it should do?
- Do input data types support intended function?
- Edge cases: what are the min/max ranges?

Negative test cases check that bad things do not happen.

- Confirm necessary validations are present
- Do unexpected inputs cause unexpected results?
  - Pressing ENTER without input
  - Over-the-Edge cases: LT min, GT max.

<u>Black Box Testing</u>: Compile the source code in BlackBox-StringDemo.c (cpr101.ca)

Intended function of the program:

- Prompt the user for a string of characters,
  - then (re)prompt to extract the character from a position within the string.
- A position input of zero will end that prompting loop and return to prompt for a new string.
- Input of an empty string ends the program.

```
*** Black Box testing: extract character from string ***
Enter a string of characters or [ ENTER ] to end.
----+----1----+----2----+----3----+----4
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Enter the character position to extract or zero to end.
> 1
  is 'a'
> 31
  is 'a'
[Which 'a'?]
```

```
*** Black Box testing: extract character from string ***
Enter a string of characters or [ ENTER ] to end.
----+----1----+----2----+----3----+----4
1234567890
Enter the character position to extract or zero to end.
> 1
  is '1'
[Would '1' be the character in that position or the position (index) itself?]
```

```
*** Black Box testing: extract character from string ***
Enter a string of characters or [ ENTER ] to end.
----+----1----+----2----+----3----+----4
12321
Enter the character position to extract or zero to end.
> 1
  is '3'
Enter the character position to extract or zero to end.
> 2
  is '2'
Enter the character position to extract or zero to end.
> 3
  is '1'
```

The samples above in no way verify the accuracy of the underlying software.

CREATE TEST CASES TO ILLUSTRATE WHETHER THE PROGRAM WORKS AS A USER WOULD REASONABLY EXPECT.

- Positive test cases should always pass
- Negative test cases should pass
    - but may fail if the programmer did not do
        - sufficient validation or
        - made assumptions about the quantity or quality of input data

Experimentation with a program is usually done before documenting test cases especially in the case of black box testing. Exhaustive testing need not be recorded.

Test cases illustrate:

- Did the programmer understand the program's "intended function" specifications correctly?
- Is the user interface consistent in what it suggests the user should do versus what the user can do? How does the UI take care of the user if they make a mistake?
- What quality of test data should be used?
  N.B. Repeating sequences of values should never be used.
- What data will verify the program generates correct output?

Using a typical length string, develop **test case data and values for positions:**

- verify the minimum edge condition: Positive case, 1=first position of string
- verify the maximum edge condition: Positive case, *n*=last position of string
- verify nominal behaviour: Positive case, minimum < **nominal** < maximum
- verify repeatability. Just because it works the first time does not necessarily mean it works the next time. (Never assume a program function's initial state will [not] continue to be its ongoing operational state.)
- verify beyond the minimum edge condition: Negative case, first position less 1, less 2
- verify beyond the maximum edge condition: Negative case, last position plus 1, plus 2
- Negative tests for unexpected input:
    - What happens when an atypical length string is used? (< minimum, > maximum)
    - What happens if something other than the expected data type is input, e.g. a whole number (integer) for the character position?

Does the interface accurately translate 1=first, 2=second, 3=third, … requested character positions to the program's internal representation of the string? Based on your knowledge of C string processing, what is the bug and how should the programmer correct it?

Does the program work accurately with different length strings? What is the minimum length string? the maximum length?

Are there constraints on the type of inputs accepted? i.e. what works? What doesn't? What happens if something other than a whole number is input for the character position?

What validation should the program perform to avoid input problems or incorrect output?

White Box testing of a Tic-Tack-Toe game (AKA noughts and crosses or Xs and Os)

See the code in WhiteBox-TicTackToe.c (cpr101.ca) and compile it.

The program provides a playing grid and alternates turns between two players who make their mark by entering a position number. There is logic recognize wins. Strategy is up to the players.

N.B. This program allows players to NOT follow the normal rules of this game!

Considerations for White Box testing start with the same concerns and strategies as for Black Box testing. See the slides in the lecture presentation for additional testing opportunities that seeing the source code allows. The obvious opportunity is all lines of code can be examined for testing.

What are the test cases to show how a player is permitted to misbehave to gain an advantage?

What are the test cases to show how a player may be disadvantaged by the program's logic?

The program identifies a win by traditional metrics *and more*. What are the test cases to show what this game allows as a single win, multiple wins, and the game winning condition?

Knowing how scanf() interacts with the OS stdin buffer and when the OS interacts with the user is the key to testing how one player can control the other player's moves.

Test Case table

Description	+ / − Purpose	Data Input	Expected Output	Actual output if unexpected	Success?	Comments

Individual test case components

Description:

- The reason and intent of each test; this is like the rational for source code comments.

Positive or Negative Test and its Purpose:

- **"+"** indicates a positive case which expects a [PASS] result for inputs in normal range of use and to illustrate minimum & maximum values that can be processed, i.e. up to the edge cases.
- "−" indicates a negative case to generate a validation message or error handling [PASS] or to explore potentially unexpected / undefined behaviours beyond the edge cases [FAIL].

Data Input:

- exact value or instructions to create input that illustrates Description and Purpose.
- It specifies exactly what and how data will be input but not why. Why is in the Description.

Expected Output:

- reference value to confirm the test result – this is documented before the test is run.

Actual result if unexpected:

- any variance from the expected output

Success?

- "PASS" or "FAIL"

Comments:

- required for a test case that FAILs and/or for unexpected results.
- If the test fails, recommend a fix to prevent the failure, e.g. a validation check and/or diagnostic message.
- If the test passes, comments are needed only to clarify assumptions, constraints, or special conditions required for the input to result in the expected output.

Useful test string of unique characters

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%^&*()_+-=[]\{}|<>?/;'":
```