5. For loop

What happened to the snippet?

```
1 #include <iostream>
2 using namespace std;
3 int main(){
4   for (; ;) //
5     cout << "Please press Ctrl + c to stop me!" << endl;
6 }
```

same as while(true)

6. Random numbers with demos

6.1 **int rand(void)**: returns a pseudo-random number in the range of 0 to RAND_MAX (). RAND_MAX is a constant defined by the implementation.

6.2 ***void srand(unsigned seed)***: Seeds the pseudo-random number generator used by rand() with the value seed. Usually, a parameter, **time(0)**, is used for **srand(). time()** returns a time_t value which vary every time and hence the pseudo-random number vary for every program call.

varies everytime

Example1:

```
1 #include <cstdlib>
2 #include <iostream>
3 using namespace std;
4 int main()
5 {
6   for(int i = 0; i<5; i++)
7     cout << rand () << endl;
8 }
```

five items

run this again, the results will be the same since there is an assigned _____ ?

Example2:

```
1 #include <cstdlib>
2 #include <ctime>
3 #include <iostream>
4 using namespace std;
5 int main()
6 {
7    srand((unsigned)time(0));
8    for( int i = 0; i < 5;i++ )
9    {
10   cout << rand() << endl;
11   }
12 }
```

necessary

makes answers vary each time

Example 3(Generating a number in a specific range)

```
1 #include <cstdlib>
2 #include <ctime>
3 #include <iostream>
4 using namespace std;
5 int main()
6 {
7    srand((unsigned)time(0));
8    for(int index=0; index<5; index++)
9       cout << (rand()%10)+1<< endl; //
10 }
```

[m,n]

o/o (n-m+1)+m

~ %10

[0,9] +1

⇓

[1,10]

Maggie Shipman

Chapter2 *COMP2710* *Xuechao Li*

1. Black box test—Equivalence Partitioning

Basic idea: consider input/output domains and partition them into equiv. classes

Scenario: Search for a value in an array: Return value is the index of some occurrence of the value, or -1 if the value does not occur in the array.

+5

1.1 Please list all possible partitions with examples to test if a "search" algorithm can work for all cases.

1. input : Array + target value (T.V)
   - empty array → -1
   - or non-empty array { single element / more than one element
   - A: (1, -2, 3, -4 5, 6, 0)

2. -1 / location of TV { 1st / last / middle

3.

Array	T.V.	output
Empty	+5	-1
[5]	5	0
[-5]	5	-1
middle [1,-2,3,-9,6,5,0]	5	5
last [1, -2,3,-4,5]	5	4
first [5, 1, -2,-3]	5	0

2. White box test

For a given snippet, please specify values for all possible variables to achieve both coverages.

branch 3 statement ~~systems~~

```
if(a || b)) {
    test1 = true;
}
else {
  if(c) {
    test2 = true
  }
}
```

statement → test1 = true;

statement → test2 = true

1. statement coverage

Test #1: T } a, F } T — b: T F — c: x matter

Test #2: F — T or F — T

2. branch coverage

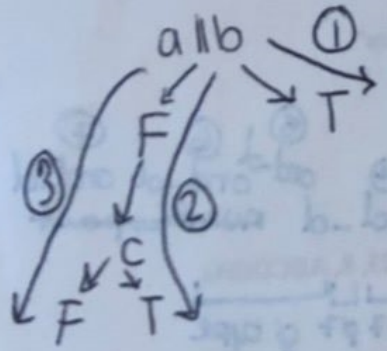
inputs: a, b, c

a	b	c
F	F	F

→ terminate prog !!!

test 2 = true

test 1 = true



1. precision

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5 double a = 30;
6 double b = 10000.0;
7 double pi = 3.1415926;
8 double pi_round = 3.14151926;
9
10 cout.precision (5);
11
12 cout << showpoint << "a1 = " << a << '\t' <<"b1 = " << b << '\t' << "pi1 = " << pi << '\t' << "pi_round1 = "
<< pi_round <<'\n';
13 cout << noshowpoint << "a2 = " << a << '\t' <<"b2 = " << b << '\t' << "pi2 = " << pi << '\t' << "pi_round2 =
" << pi_round <<'\n';
14 cout << fixed<< "a3 = " << a << '\t' <<"b3 = " << b << '\t' << "pi3 = " << pi << '\t' << "pi_round3 = " <<
pi_round <<'\n';
15 }
```

↓ wants 5 digits

ø want decimal

wants 5 digits after decimal

5 or 6 displayed?

a1 = 30.000 b1 = 10000 do you have to display decimal pi1 = ~~3.1415926~~ 3.1416 au compiler rounds it. pi_round1 = 3.1415 ~~1926~~

a2 = 30 b2 = 10000 pi2 = 3.1416 bc compiler maintains correctness pi_round2 = 3.1415

a3 = 30.00000 b3 = 10000.00000 pi3 = 3.14159 pi_round3 = 3.14152

2. Switch with continue and/or break

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4 while(true)
5 {
6  int i;
7  cin >> i;
8  switch( i )
9  {
10    case 1:
11    case 2:
12    case 3:
13        cout << "inside switch " << i << "\n";
14        continue;
15    default:
16        cout << "inside default\n";
17        break;
18 }
19 cout << "Bottom of while " << i << endl;
20
21 }
22 }
```

output if i=1
inside switch 1

output if i=∅
inside default
Bottom of while ∅

move back to while

< bc no break or continue, step to next case

not a switch word, a while command

i=∅ goes straight here

compatible w/ switch

jumps out of switch

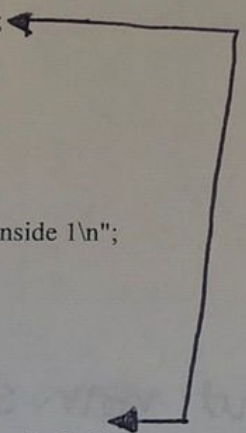
end of line, if there is extra input it will be put on new line

> if i = 1,

if i = 0,

3. Does the following sample code works? If yes, please predict outputs. If no, explain it.

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6  int choice = 2 ;
7  switch(choice)
8  {
9   case 1 :
10        cout << "inside 1\n";
11        break;
12
13   case 2 :
14        cout << "inside 2\n";
15        continue;
16
17   case 3 :
18        cout << "inside 3\n";
19        break;
20 }
21 }
```



compiling bug, continue bad w/ switch

can only use break w/ switch case

4. If….(else)……

Please predict outputs

```
1 #include <iostream>
2 using namespace std;
3 int main(){
4   int num=70;
5   if( num < 100 );
6     /* This cout statement will only execute,
7      * if the above condition is true
8      */
9     cout<<"number is less than 100" << endl;
10
11
12  if(num > 100);
13    /* This cout statement will only execute,
14     * if the above condition is true
15     */
16    cout<<"number is greater than 100" << endl;
17
18  return 0;
19 }
```

output: both statements

logical error

Example2:

```
1 #include <cstdlib>
2 #include <
3 #inclu
4 usin
5 int
6 {
7
8
9
10
11
```


3. A Test Driven Development Example

Assume that we are building a new application, and one of the modules will provide message processing. We are not sure of the complete functionality needed, but we are sure that we need to be able to send and receive messages, and further, that the messages will consist of a variable number of 32 bit integer values. We will first create a simple C++ header file and define a procedure to send messages, and a procedure to receive messages. It might look like the following:

```
// Message.h
bool Send_Message(MessageIdType Message_Id, ByteStream &MessageData, MessageSizeType MessageSize);
bool Receiv_Message(MessageIdType &Message_Id, ByteStream &MessageData, MessageSizeType &MessageSize);
```

Now rather than go forward with the implementation details of these procedures as the next step, please start a brain storm with your friends to generate test cases for these two procedures. List design details and the edge conditions before we implement our test cases.

① invalid msg ID #'s ② sizeof msgp ③ format → msg { text mm msg

4. Syntax test

The Backus-Naur Form (BNF) is a way of defining syntax. It consists of:

- a set of terminal symbols
- a set of non-terminal symbols
- a set of production rules of the form
- Left-Hand-Side(LHS) ::= Right-Hand-Side(RHS)

ordinary-digit ::= 0 1 | 2 3 4 5 6 7 8 9

As a software engineer, you are required to implement partial of a cell phone app to detect if phone numbers entered by users are correct.

# of returns

1	special_digit	::= 0 | 1	::= means "is defined as".
[	other_digit	::= 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9	| means "or".
1	ordinary_digit	::= special_digit | other_digit (0 1 | 2 3 4 5 6 7 8 9)	* means "zero or more occurrences".
3	exchange_part	::= other_digit^2 ordinary_digit	+ means "one or more occurrences".
4	number_part	::= ordinary_digit^4	A^n means "n repetitions of A".
7	phone_number	::= exchange_part number_part	

For example: phone # ::= other_d ① other_d ② ord_d ③ ord_d ④ ord_d ⑤ ord_d ⑥ ord_d ⑦ number_part

Correct phone numbers: 3469900, 9904567, 3300000

Incorrect phone numbers: 0551212, 123, 8, ABCDEFG

7 values
1st 2 digits 1/0 (not)

∅7 ∅7 ∅ type
long

Exam

1

4.1 Please develop your test cases to expose any vulnerabilities of this cell-phone app for the following four BNF forms.

phone_number ::= exchange_part number_part

case 1: empty → fail
case 2: exch_p exch_p or num_p num_p → fail
case 3: numb_p exch_p → fail

exchange_part ::= other_digit^2 ordinary_digit

number_part ::= ordinary_digit

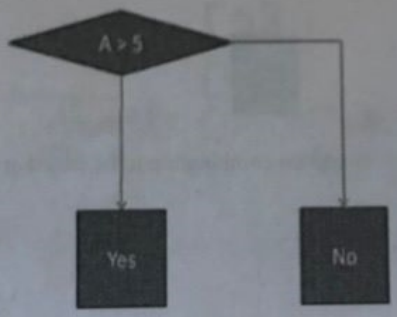
ordinary_digit ::= special_digit | other_digit

While box test.

1. Statement coverage: every possible statement in the code to be tested at least once.

what is the difference btw statements vs decision points

2. Decision coverage: the true or false outcomes of each decision point to be tested at least once.



test case generate 2 outcomes: T ; F

"big black box testing"

3. Path coverage: All paths to be tested at least once. ASK?

Xuechao Li

Maggie Shipman

Chapter2 COMP2710

Midterm Practice *COMP2710* *Xuechao Li*

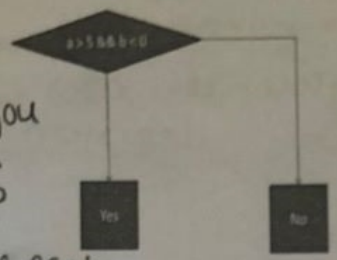
4. Condition coverage: the true or false outcomes of each Boolean expression to be tested at least once.

① a=6
a>5 → T
b<0 → F

b=6
a=-6

② a>5 → F
b<0 → T b=-6 once each

all you need a 3 b T & F



generate T ⅀ F

condition: each part of boolean could return F for whole thing ⅀ still testing correctly

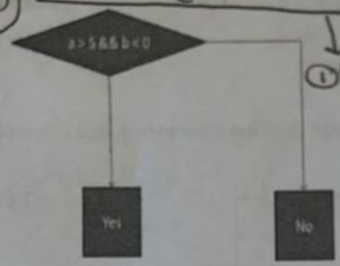
5. Branch coverage: Decision coverage + Condition coverage (T ⅀ F from each Bool)

③ a=6
a>5 → T
b<0 → T
b=-6

✓ (T ⅀ F)

① ③
or
② ③

① ②



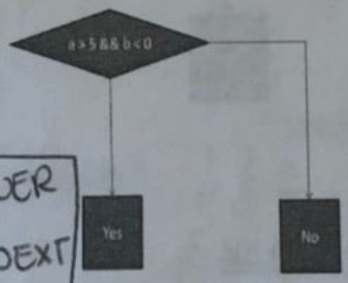
Need 1,2,3 for Branch Coverage

decision: "big black box" entire decision return "big box" T ⅀ F once

6. Branch condition combination coverage: All possibilities of condition combination to be tested at least once.

1. a>5 → T: 1.1, F: 1.2

2. b<0 → T: 2.1, F: 2.2



1.1 2.1 | 1.2 2.1 | 1.1 2.2 | 1.2 2.2

TO GIVE ANSWER WRITE PATH ⅀ TEST CASE NEXT TO IT

all 4 cases we only missed F-F case

★ 2 # of boolean expression in each decision point

ask!

7. Truth table.

Group A: (1) ¬(p ∧ q) (2) ¬p ∨ ¬q

p	q	¬p	¬q	p && q	¬p || ¬q
T	T	F	F	T	F
T	F	F	T	F	T
F	T	T	F	F	T
F	F	T	T	F	T

∨ "or"

∧ "and"

¬ "NOT"

Review Comp 2710

std namespace

Q1. 57 call-by-value/reference

```
main ()                int fool (a, b)   <- call by value
{                      int fool (&a, &b) <- call by ref
   fool (1 2)          1 diff 2
}
```

A 1 B 2 C 3 D 4

6) Iteration { for, while, do while } { if... else..., switch }

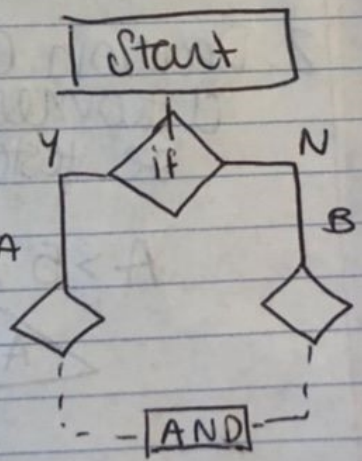
Q2. While or do while convert to for loop(; ;)

Q3. if... else if ... else w/ enumeration

convert to

switch case

```
enum Genders {
        MALE
        FEMALE
             }
```



Q4. white box testing
- given flow chart

design test case for each
1. statement coverage
2. decision coverage
3. condition coverage
4. branch coverage
5. branch condition combination coverage
6. path coverage

Testcase: if ... else ...

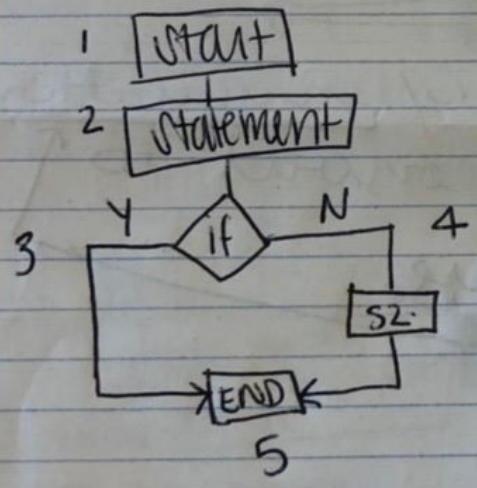
Xuechao Li

Maggie Shiemmin

Q5 Black Box Test
ex phone number system
develop proper test cases for each sub-components of a proposed system

Worksheet:

White Box Test:
1. statement coverage: every possible statement in the code to be tested @ least once



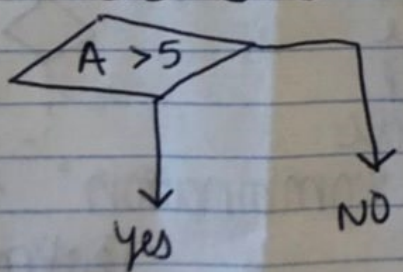
how many statements:
S1, S2 ?

Write out all paths

2. Decision Coverage: the true or false outcomes of each decision point to be tested @ least once

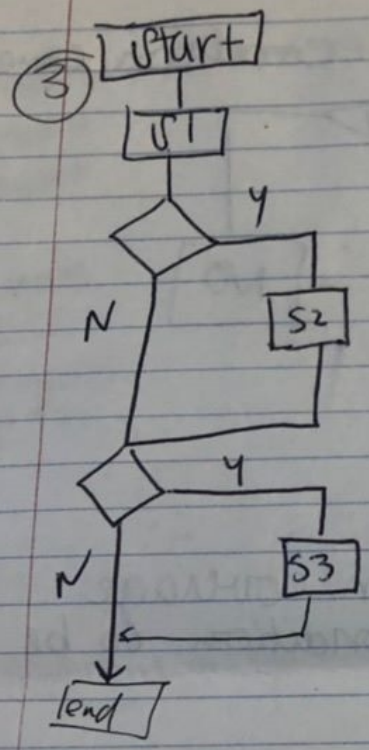
this is the work

A>5 && C>0



A	T	7		F	3
	+			+	
C	T	1		T	1
	⇩			⇩	
	T	T		F	F

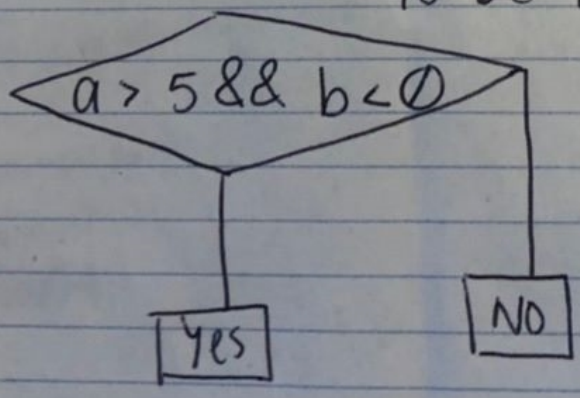
COMP2710 Maggie Shipman Xuechao Li



Path Coverage:
All paths to be tested at least once

write all paths and test cases that cause you to go down each path once

Condition Coverage:
the true or false outcomes of each boolean expression to be tested @ least once

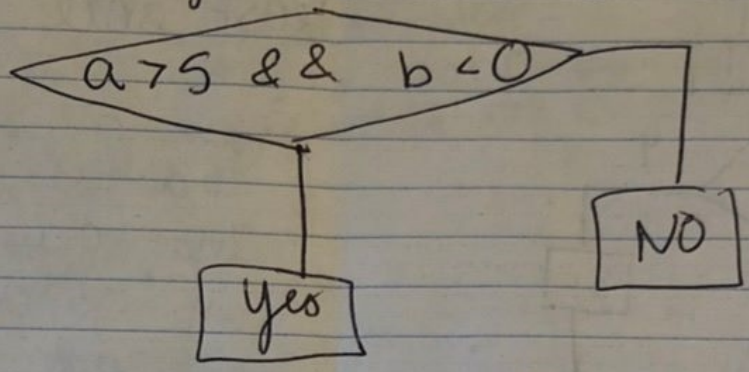


	T	F
1. a > 5 && b >= 0	↓ 6	↓ 5
	F	T
2. a <= 5 && b < 0	↓ 4	↓ -4

5. Branch coverage : Decision + Condition coverage

T { a 6 ← T
  { b -1 ← T

F { a 4 ← F
  { b 4 ← F



6. Branch condition combination coverage:
All possibilities of a condition to be tested at least onke