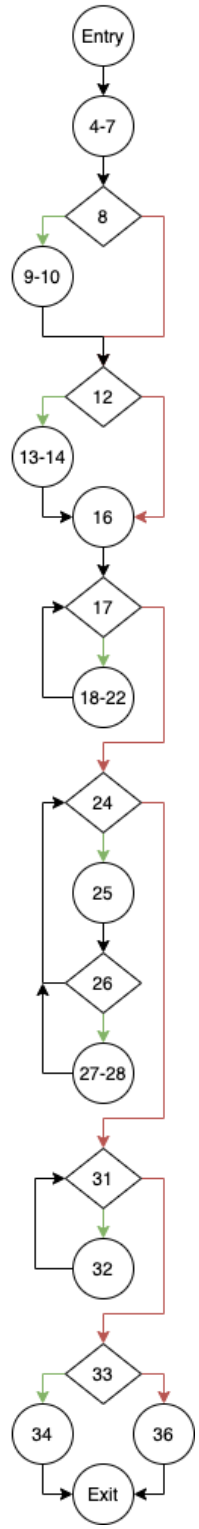
CSCI 3100 Software Engineering

Assignment 6

Jonathan Lam Pui Hong (1155093597)

Q1 Figure



Q1.1

Refer to figure on the left. Nodes are labelled with their corresponding line number(s), excluding quotations ({}) and edge-related key words (else).

Q1.2

Nodes	Edges	Predicate nodes	Regions	V(G)
19	25	7	8	8

Q1.3 V(G) = Edges - Nodes + 2 = Regions = Predicate nodes + 1

Q1.4

Entry, 8, 12, 16, 17, 24, 31, 33, 36, Exit
Entry, 8, 12, 16, 17, 24, 31, 33, 34, Exit
Entry, 8, 12, 16, 17, 24, 31, 32, 31, 33, 36, Exit
Entry, 8, 12, 16, 17, 24, 25, 26, 27-28, 24, 31, 33, 36, Exit
Entry, 8, 12, 16, 17, 24, 25, 26, 31, 33, 36, Exit
Entry, 8, 12, 16, 17, 18-22, 17, 24, 31, 33, 36, Exit
Entry, 8, 12, 13-14, 16, 17, 24, 31, 33, 36, Exit
Entry, 8, 9-10, 12, 16, 17, 24, 31, 33, 36, Exit

Q2.1 Yes. {< a = -2, b = -2 >, < a = -2, b = 2 >}

Q2.2

No. An additional test < a = 2, b = 2 > is needed to cover the edges from 8 to 12 and 12 to 16.

Q2.3

Yes. {< a = -2, b = -2 >, < a = -2, b = 2 >, < a = 2, b = 2 >}.
Yes, it is the same. C_1 covers all edges in the statement. Since there is no compound condition (with AND) but only nested condition (with OR, in 17), C_2 and C_1 is the same.

Q2.4

No, kind of. The 3 loops with the addition of a few if-then(-else) statements make the number of test cases needed for path coverage exceedingly large.

Q3.1 & 3.2

This program tries to run an addition on two (32-bit) integers.

The program first converts two (decimal) numbers into an array with the first bit as a flag of their signs (1: negative, 0: positive) and the remaining digits as their binary representation. It then adds the two arrays together, with carrying where necessary, and coverts it back to integer. It finally checks the first bit to determine the sign of the number.

1. Negative addition:
   a. Sum error: When two negative numbers are added, since signs are not checked, it performs addition on the numbers as if they are positive.
   b. Sign error: The sign of the result is "times together" rather than "add together", i.e., if two negative numbers are added together, the sum will have a positive sign; if one -ve one +ve, the sum will have a negative sign. (This has not counted the case where there is an overflow error, e.g., test(-2147483647, 1) returns 0. See below.)
2. Integer overflow:
   a. -214783648 as zero: Although -214783648 is counted as negative in C++ integer, the program counts it as 100…[27x]...00, i.e. negative zero.
   b. Sum overflow (and syntax error): The program's signed integer (32-bit, with two signed zeros) can only accept numbers from -2147483647 to 2147483647. If the sum of two numbers is larger than this range, error occurs. A corresponding syntax error also happens in line 28 where a negative index (-1) of array may be accessed.

Q3.3

Test case		Example		Result		
		a	b	c	Valid	Bug
Zero	Zero parsing	0	0	0	T	
	Zero-positive addition	0	2	2	T	
	Zero-negative addition	0	-2	-2	T	
Position	Input position	2	0	2	T	
In-range	Pos-positive addition	3	2	5	T	
	Neg-positive addition	-3	2	-5		1a
	Neg-negative addition	-3	-2	5		1b
Edge	Edge case (negative)	0	-2147483648	0		2a
	Edge case (positive)	0	2147483647	2147483647	T	
Overflow	Overflow case (negative)	1	-2147483648	-1		2b
	Overflow case (negative)	1	-2147483647	0		
	Overflow case (positive)	1	2147483647	0		

Remarks: Edge or overflow cases use INT_MIN and INT_MAX of C++, -2147483648 and 2147483647 respectively. Since "Edge case (negative)" detects the problem of INT_MIN, the "overflow case (negative)" are modified accordingly.

Q3.4

For white-box testing, no. White box testing can only make test cases that pertains to the logic flow and program paths, in which the bugs found may not be the concern of the purpose of the program. For black-box testing, yes. If the test cases are well-designed, the bugs can be found. (Corresponding bugs are marked on the rightmost column in previous table.)

Q4.1

Not related

GUI module, Live streaming module

Stub module

Simulated *transaction module*: The module which has the same API called by the *income calculation module* can validate the amount of money and the bank information of the account. (Then, if applicable, it will transfer the money to the said account.)

```
procedure transaction(content, account) do
    print "Money sending: "
    print "${content.currency} ${content.income}"
    print "Account receiving: "
    print account.bank
    // transferMoney();
end procedure
```

Simulated *emailing module*: The module which has the same API called by the *income calculation module* can generate a confirmation letter and validate the letter and the email account. (Then, if applicable, it will send the email to the said mail account.)

```
procedure emailing(content, account) do
    print "Sample email: "
    print "${content.currency} ${content.income} is sent to ${account.bank}"
    print "Mailing account: "
    print account.email
    // sendEmail();
end procedure
```

Driver module

Simulated *data recording module*: The module can generate (or allow an input of) a data object of a stream and call the *income calculation module* with this data.

```
procedure dataRecording()
    // streamData = fetchDataFromStream(streamID);
    print "Data from live streaming: "
    input streamData // Temporary input to be changed
    incomeCalculation(data, _);
end procedure
```

Simulated *account management module*: The module can generate (or allow an input of) a account object and call the *income calculation module* with this account.

```
procedure accountManagement() do
    // account = fetchAccountFromDB(streamID);
    print "This is the account: "
    input account // Temporary input to be changed
    incomeCalculation(_, account);
end procedure
```

Remarks: Types/Structures used

```
struct bankAccount do
    accName: string;
    accNum:  string;
    accBank: string;
end struct

struct account do
    username: string;
    name: string;
    bank: bankAccount;
    email: string;
end struct

struct content do
    money: float;
    currency: string;
end struct
```

Q4.2

Number of customers	Gift value	Expected income
8,256	520	424.00
12,863	2,753	1,357.20
125,742	8,641	5,538.35
1,246,732	12,845	28,787.50

Q4.3

Given the clarification in Piazza, if x is the number of customer,

$$x \in [0\,..\,8546] \cup [10^4\,..\,91409] \cup [10^5\,..\,905999] \cup [10^6\,..\,\infty).$$