Assignment 1, 2020-09-14

Question 1 (Bitwise Operations). Write the output (and the content of variables `a,b,c` in hexadecimal notation), after this snipped is executed:

```
#include <iostream>
using namespace std;
int main() {
    unsigned int a =
      0xACE02468;
    unsigned int b =
      (a << 12) & (a >> 20);
    unsigned int c =
      (a << 12) | (a >> 20);
    cout << hex <<
      "a = " << a << endl;
    cout << hex <<
      "b = " << b << endl;
    cout << hex <<
      "c = " << c << endl;
}
```

Hexadecimal memory content of variables:

Variable	Hex value
`a`	
`b`	
`c`	

Note. Unsigned ints are 4 bytes long. If you do a right shift on such variables (by some n bits), then the first n bits on the left are filled with zeroes.

Question 2. Draw a flowchart for this `switch-case` statement.

```
int x = 0;
char c;
cin >> c;
switch( c ) {
    case 'A':
        x += 1;
    case 'B':
        x += 2;
        break;
    default :
        x += 4;
}
cout << "x= " << x << endl;
```

Use only 5 kinds of nodes:
(1) Start node (oval: one outgoing arrow).
(2) End node (oval: one incoming arrow).
(3) Conditional statement (diamond: one incoming and two outgoing arrows). Mark the "true" branch.
(4) Regular statement (rectangle: one incoming and one outgoing arrow).
(5) Merging two branches (black dot: two incoming arrows, one outgoing arrow).

Question 3 (Side Effects). What is the value of `x` output by the code snippet above, if `cin` inputs letter `'A'`?

Solutions

Question 1 (Bitwise Operations).

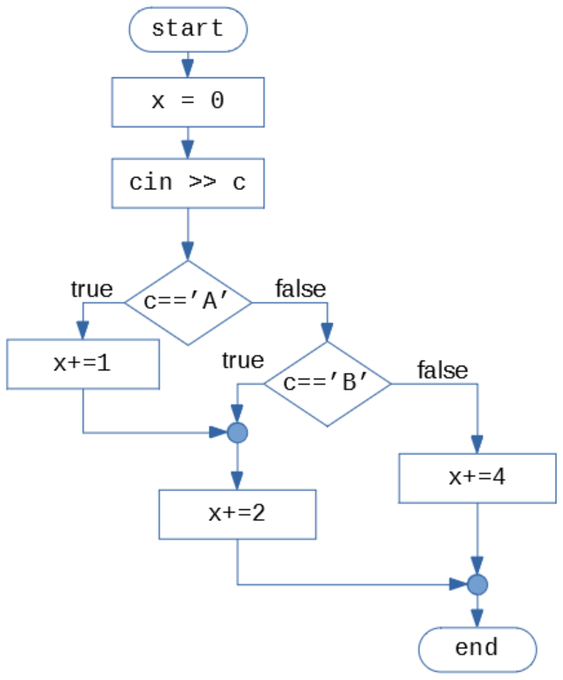
Variable	Hex value
a	ACE02468
b	00000000
c	02468ACE

Standard output from the program looks like this:

```
$ ./myprogram
a = ace02468
b = 0
c = 2468ace
```

Variable *b* is filled with 0s, because bitwise-AND (written as `&` in the expression `(a <<12) & (a >>20)`) is run on two expressions that do not have 1-bit in the same place. If we shift any number left by 12, then its last 12 bits are filled with 0-bits. If we shift any number right by 20, then its last 20 bits are filled with 0-bits. Variable *c* has the same number of 1-bits as *a*, but its bits are rotated (the first 12 bits travel to the end of the variable).

Question 2 (Flowchart).
Switch statement is similar to any other conditional (in certain situations it is more efficient than if/else statements with many branches). The intersting thing about this flowchart is missing (forgotten?) `break` statement after Line 6 in the source code. If the input char equals to `'A'`, then we run code for **both** branches –it also runs the increment that is under the branch `'B'`.



Question 3 (Side Effects). Variable `x` has value 3 - initially it is 0, but it is incremented by 1, then by 2 in two different case statements (notice, there is no `break` after the first `case`).