CS1010E: Programming Methodology

Tutorial 03: Selection

06 Feb 2017 - 10 Feb 2017

1. Discussion Questions

(a) [Bad Practices] What is/are the output of *badly written* code fragments below?

i.
```c
int a = 3, b = 4, ans = 1;
if(b > a > 1) {
  ans = a + b;
} else {
  ans = a - b;
}
printf("%d", ans);
```

i. ______________________________

ii.
```c
int a = 3, b = 4, ans = 1;
if(a > 3)
  ans  = a + b; ans += 1;
printf("%d", ans);
```

ii. ______________________________

iii.
```c
int a = 3, b = 4, ans = 1;
if(a >= 3) ans = a + b; else ans = a - b;
printf("%d", ans);
```

iii. ______________________________

iv.
```c
int a = 3, b = 4, ans = 1;
if(a >= 3)
  if(b > 4)
    ans = a + b;
else
  ans = a - b;
printf("%d", ans);
```

iv. ______________________________

v.
```c
int a = 2, ans = 1;
if(a%2) ans = 0;
printf("%d %d", a, a%2);
```

v. ______________________________

2. Program Analysis

(a) [Operation Precedence] What is/are the output of code fragments below?

i.
```
int x = 5, y = 8, z = 13, ans;
ans = x + y < z ? ++x < y ? x++ : y++ : x < z ? x++ : z++;
//  = ( (x + y < z ) ? ( (++x < y) ? x++ : y++ ) : ( (x < z ) ? x++ : z++ ) )
//  = ( (5 + 8 < 13) ? ( (++x < y) ? x++ : y++ ) : ( (x < z ) ? x++ : z++ ) )
//  = ( (   13 < 13) ? ( (++x < y) ? x++ : y++ ) : ( (x < z ) ? x++ : z++ ) )
//  = ( false        ? ( (++x < y) ? x++ : y++ ) : ( (x < z ) ? x++ : z++ ) )
//  =                                              ( (5 < 13) ? x++ : z++ )
//  =                                              ( true     ? x++ : z++ )
//  =                                                           5        [x = 6]
//  [x = 6 ("by increment"), y = 8 ("unchanged"), z = 13 ("unchanged"), ans = 5]
printf("%d %d %d %d", x, y, z, ans);
```
i. 6 8 13 5

ii.
```
int x = 0, ans = 0; // follow the step (#) for execution path
switch(x) {      // (1) x == 0
  case 1:  ans = 1;   x++;
  // (2) SKIP case 1 because x != 1
  case 0:
  // (3) INTO case 0 because x == 0
    ans += 2;   // (4) ans = ans + 2 = 0 + 2 = 2
    ++x;        // (5) [x = 1] ("by increment")
  case 2:
  // (6) FLOW into case 2 because no "break" in case 0
    if(x == 0) break; // (7) NOT executed since x == 1 now
    ans++;            // (8) [ans = 3] ("by increment")
  case 3:
  // (9) FLOW into case 2 because no "break" in case 2
     ans -= 4;        // (10) ans = ans - 4 = 3 - 4 = -1
     break;           // (11) EXIT from switch
  default: ans = 0
  // (12) SKIP default because "break" above
} printf("%d", ans);  // [ans = -1]
```
ii. -1 (case 0, 2, 3 executed)

iii.
```
int a = 1, b = -1, c = 0, d = -1, e = 0, f = 1, ans;
ans = --a    ||  b++  && c--   && ++d  || ++e  || --f;
//  = --a    || (b++  && c--   && ++d) || ++e  || --f;
//  = 0      || (b++  && c--   && ++d) || ++e  || --f; [a =  0]
//  = false || (b++  && c--   && ++d) || ++e  || --f; [a =  0]
//  =          (-1   && c--   && ++d) || ++e  || --f; [b =  0]
//  =          (true && c--   && ++d) || ++e  || --f; [b =  0]
//  =          (        0     && ++d) || ++e  || --f; [c = -1]
//  =          (        false && ++d) || ++e  || --f; [c = -1]
//  =          ( false               ) || ++e  || --f; [c = -1]
//  =                                    1    || --f; [e =  1]
//  =                                    true || --f; [e =  1]
//  = true = 1 ("by type conversion from boolean to int")
// [a = 0, b = 0, c = -1, d unchanged, e = 1, f unchanged, ans = 1]
// NOTE: Conversion Values
//  :: 0     --> false | any other number --> true [int -> bool]
//  :: false --> 0     | true             --> 1    [bool -> int]
printf("%d %d %d %d %d %d %d", a, b, c, d, e, f, ans);
```

iii. `0 0 -1 -1 1 1 1`

(b) [Abstraction] State (*in English*), what is the purpose of the following code fragments below?

i.
```
// given a, b, c as user input
if(a > b)   // a > b
  if(a > c) ans = a; // a > b  && a > c                       --> a is MAX
  else      ans = c; // a > b  && c >= a --> c >= a >  b --> c is MAX
else        // b >= a
  if(b > c) ans = b; // b >= a && b > c                       --> b is MAX
  else      ans = c; // b >= a && c >= b --> c >= b >= a --> c is MAX
printf("%d", ans);   // ans is the MAX value between a, b, and c
```

i. `finding maximum of three numbers`

ii.
```
// given x as user input
if(x < 0)           // x    <  0
  ans = -x;         // ans  >= 0  --> ans is positive
else                // x    >= 0
  ans = x;          // ans  >= 0  --> ans is positive
printf("%d", ans);  // ans is always positive
```

ii. `finding the absolute value`

(c) [Limit of Values] What is/are the output of code fragments below?

i.
```
if(0.7 == 0.3 + 0.4)
  printf("Equal");
else
  printf("Not Equal");
```

i. `Not Equal (imprecision)`

3. Designing a Solution

(a) [Computation] Given that the 1st of January, 2017 falls on a Sunday, determine the day of the week for any given date in 2016. Note that it is possible to be done without using any repetition construct (*which some of you may not have yet learned*). Write your program below:

```c
#define SUN 0   #define MON 1   #define TUE 2   #define WED 3
#define THU 4   #define FRI 5   #define SAT 6
#define JAN 1   #define FEB 2   #define MAR 3   #define APR 4
#define MAY 5   #define JUN 6   #define JUL 7   #define AUG 8
#define SEP 9   #define OCT 10  #define NOV 11  #define DEC 12

int main() {
  int day, month, week = 0; scanf("%d %d", &day, &month);
   /* Compute Day of the Week */
   switch(month) { // Add the number of days from January
     case DEC: day += 30; // November  added
     case NOV: day += 31; // October   added
     case OCT: day += 30; // September added
     case SEP: day += 31; // August    added
     case AUG: day += 31; // July      added
     case JUL: day += 30; // June      added
     case JUN: day += 31; // May       added
     case MAY: day += 30; // April     added
     case APR: day += 31; // March     added
     case MAR: day += 29; // February  added (also, leap year)
     case FEB: day += 31; // January   added
     case JAN: day = (day + 6) % 7; // No addition: perform computation
   }
  switch(week) {
    case SUN: printf("Sunday");    break;
    case MON: printf("Monday");    break;
    case TUE: printf("Tuesday");   break;
    case WED: printf("Wednesday"); break;
    case THU: printf("Thursday");  break;
    case FRI: printf("Friday");    break;
    case SAT: printf("Saturday");  break;
  }
  return 0;
}
```

(b) [Boolean Logic] The Gregorian calendar has a pretty complicated rule for determining leap year. The rule that says "year that is divisible by 4" is in fact, *incomplete*. The exception to this rule is that for years that are divisible by 100, it is a leap year only if it is also divisible by 400. Otherwise, it is considered a common year.

Given the description above, the rule for leap year can be summarized as follows for any given `year`:

1. `year` that is divisible by 4 is leap year, *except for*
2. `year` that is divisible by 100 –which is common year– *unless it is*
3. `year` that is divisible by 400 –which is leap year.

Write the code to determine if a given `year` is a leap year or not. You are given two sets of template below, answer for both cases:

i. Multiple checks, *using **only**[1] arithmetic and relational operations*:

```
int main() {
  int year; bool leap; scanf("%d", &year);
  if (      year % 400 == 0       ) {
    leap = true;
  } else if (      year % 100 == 0      ) {
    leap = false;
  } else if (       year % 4 == 0       ) {
    leap = true;
  } else {
    leap = false  ;
  }
  return 0;
}
```

ii. Single checks, *using arithmetic, relational, and **logical** operations*:

```
int main() {
  int year; bool leap; scanf("%d", &year);
  if (    year % 400 == 0 || (year %100 != 0 && year % 4 == 0)    ) {
    leap = true;
  } else {
    leap = false  ;
  }
  return 0;
}
```

4. Challenge

(a) [Flowchart] Programming is not always about writing codes, but it is always about solving problems. Flowchart is a viable option for programming as well. Diagram 1 shows a simple program for patient diagnostic. The convention for the flowchart is given below:

- Oval: Starting point
- Diamond: Decision point, user input is given via standard input
- Square: Diagnosis reached, output should be printed on standard output

Every decision has two choices: `yes` or `no`. We will represent `no` with value `0` and `yes` with value `1`. Assume that inputs are given in the following order: `fever?`, `sore throat?`, `cough?`, `chills?`, `swelling?`, and `nausea?`. Write the program corresponding to the flowchart. Write your program below:

[1]Remember that arithmetic operations involve: [`+`, `-`, `/`, `%`, etc], relational operations involve: [`==`, `>=`, `<=`, `>`, `<`, etc], and logical operations involve: [`&&`, `||`, etc]

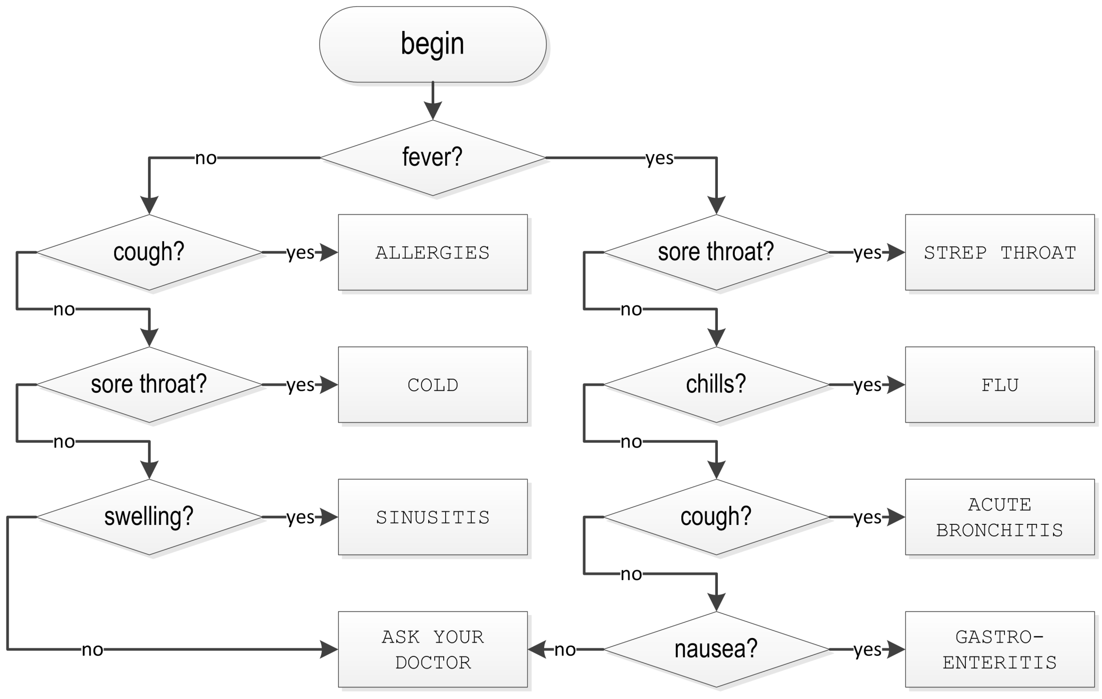

Diagram 1: A simple flowchart for patient diagnostic.

```
int main() {
  bool fever, sore, cough, chills, swelling, nausea
   /* Flowchart Program */
   scanf("%d %d %d %d %d %d", &fever, &sore, &cough, &chills, &swelling, &nausea);
   if(fever) {
     if(sore)          printf("STREP THROAT");
     else if(chills)   printf("FLU");
     else if(cough)    printf("ACUTE BRONCHITIS");
     else if(nausea)   printf("GASTRO-ENTERITIS");
     else              printf("ASK YOUR DOCTOR");
   } else {
     if(cough)         printf("ALLERGIES");
     else if(sore)     printf("COLD");
     else if(swelling) printf("SINUSITIS");
     else              printf("ASK YOUR DOCTOR");
   }
  return 0;
}
```