‘

CS 7A - Fall 2015 - Final Exam Solutions. 12/16/15

---

1. Write the number of the definition on the right next to the term it defines.

   (a) **reference** ANS:10

   (b) **variable** ANS:6

   (c) **precondition** ANS:4

   (d) **declaration** ANS:5

   (e) **parameter** ANS:1

   (f) **byte** ANS:9

   (g) **parser** ANS:8

   (h) **scope** ANS:3

   (i) **postcondition** ANS:2

   (j) **type** ANS:11

   (k) **object** ANS:7

   (1) A declaration of an explicit input to a function through which a function can access the arguments passed by name.

   (2) A condition that must hold upon exit from a piece of code, such as a function or a loop.

   (3) The region of program text (source code) in which a name can be referred to.

   (4) A requirement of a function upon its argument that must be true for the function to perform its action correctly.

   (5) The specification of a name with its type in a program.

   (6) A named object of a given type; contains a value unless uninitialized.

   (7) Some memory that holds a value of a given type.

   (8) A program that reads a stream of tokens according to a grammar.

   (9) The basic unit of addressing in most computers.

   (10) A value describing the location of a typed value in memory.

   (11) A set of possible values and a set of operations (for an object).

2. Construct `C++` loop that will

   (a) lead to an overflow error.

   ```
   for(int i = 2; i<INT_MAX; i *= i);
   ```

   (b) lead to an underflow error.

   ```
   for(double i = 2; i > i/1e10; i /= 1e10);
   ```

   (c) cause a range error.

   ```
   vector<int> vi(2); // Create the vector {0,0}.
   for(int i = 0; i <= vi.size() ; ++i) vi[i] = i;
   ```

3. Write a `C++` assignment statement that will cause a narrowing error from

   (a) an `int` to a `char`.

   ```
   int i = 321;
   char c = i; // c will be 'A' = char(321%256)
   ```

   (b) a `double` to an `int`.

   ```
   double x = 1.9;
   int i = x; // i will be 1
   ```

4. Consider the absolute value function defined below:

   ```
   double abs(double x) { return (x>0) ? x : -x; }
   ```

   Rewrite the function using an `if/else` structure instead of the ternary operator.

```
double abs(double x) {
    if(x>0) return x;
    else return -x; // else not essential here
}
```

5. The function `getPoly()` below is designed to get the coefficients, $a_0, a_1, \ldots, a_n$ of a polynomial $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$ of degree n.

```
vector<double> getPoly() {
    double deg{0};
    cout << "\nWhat is the degree of your polynomial? ";
    do {
        cin >> deg;
    } while(deg <= 0 || int(deg) != deg);
    vector<double> coeff(int(deg)+1); //create a vector deg+1 doubles = 0.0
    cout << "\nEnter the coefficients in ascending order: \n";
    for(int i = 0; i < deg+1; ++i) {
        cout << "The coefficient of x^" << i << " = ";
        cin >> coeff[i];
    }
    return coeff;
}
```

(a) What is the purpose of the `do/while` loop?// ANS: The `do/while` loop is designed to allow the user to enter a value for variable `deg` of type `double`. It then checks if `deg` is a positive integer and, if not, waits for another input without further prompting.

(b) The function could be improved by changing the type of the variable `deg`. How would you do that?
Change the declaration of `deg` to `unsigned deg;`

(c) Suppose the user wishes to enter the polynomial $x^3 - x^2$. What would the console look like when she's done?

```
What is the degree of your polynomial? 3

Enter the coefficients in ascending order:
The coefficient of x^0 = 0
The coefficient of x^1 = 0
The coefficient of x^2 = -1
The coefficient of x^3 = 1
```

(d) If each double requires 8 bytes, how many bits would the polynomial $x^3 + 2x^2 + 3x + 4$ require to store the coefficient vector?
$4\,\texttt{coefficients} \cdot 8\frac{\texttt{bytes}}{\texttt{coefficient}} \cdot 8\frac{\texttt{bits}}{\texttt{byte}} = 256$ bits.

6. The function `poly()` defined below is designed to evaluate a polynomial specified by a vector of coefficients, `coeff`.

```
double poly(vector<double> coeff, double x) {
    double value = coeff[coeff.size()-1];
    for(int i = coeff.size()-1; i > 0; --i) {
        value *= x;
        value += coeff[i-1];
    }
    return value;
}
```

Suppose `coeff` contains {0,0,-1,1} and `x = 2`.

(a) What is `coeff.size()`?
The vector contains 4 elements so `coeff.size()=4`.

(b) Complete the following table of values as the function executes until its `return` is called. Does the function return the proper value $p(2)$?

i	value
ϕ	1
3	1
	2
	1
2	1
	2
	2
1	2
	4
0	4

(c) Repeat part (b) to evaluate $p(3)$ for $p = x^3 - 2x^2 + 3$ where `coeff = {3,0,-2,1}`.

i	value
ϕ	1
3	1
	3
	1
2	1
	3
	3
1	3
	9
	12
0	12

7. Write a definition for `double secant(vector<double> coeff, double a, double b)` that will call the `poly()` function of problem #6 above and return the slope of the secant line from $(a, p(a))$ to $(b, p(b))$. You can do this in one line.

```
double secant(vector<double> coeff, double a, double b) {
    return (poly(coeff, b) - poly(coeff, a))/(b - a);
}
```

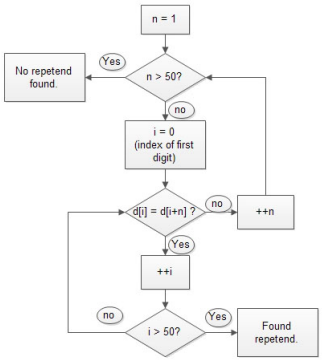
8. The decimal form of $\frac{1}{13} = 0.076923076923\cdots = 0.\overline{076923}$ has a repetend (the part of the decimal that repeats) of 076923. The program below is designed to find the length of the repetend of a fraction whose numerator is 1 and whose denominator is specified by the user. Most of the code is shown below:

```
// Note, a "short" is a 2 byte integer.
void getDecimal(vector<short>& v, unsigned n);
unsigned findRepetend(vector<short> decimals);

int main() {
    unsigned denominator;
    vector<short> decimalDigits;
    cout << "\nEnter the denominator of your rational number: ";
    while(cin >> denominator) {
        decimalDigits.clear();
        getDecimal(decimalDigits, denominator);
        cout << denominator << " has a repetend of length "
             << findRepetend(decimalDigits);
    }
}
void getDecimal(vector<short>& v, unsigned n) {
    unsigned power10 = 10;
    while(power10/n==0) power10 *= 10;
    v.push_back(power10/n);
    unsigned remainder = power10%n;
    for(int i = 0; i< 101; ++i) {
        remainder *= 10;
        v.push_back(remainder/n);
        remainder = remainder%n;
    }
}
unsigned findRepetend(vector<short> decimalDigits) {
    // supply code for algorithm
}
```

(a) Suppose `d` is a `vector<short>`. Describe what `getDecimal(d, 13)` does, step by step.
ANS:
First, `power10` is declared to be a variable of type `unsigned` initialized to 10.
Then, since `power10` ¡ 13, `power10` is multiplied by 10, yielding $100 \geq 13$, the quotient (after integer division) `power10`/13 = 100/13 = 7 is pushed onto the `vector<int>`, `v`.
Then the variable `remainder` of type `unsigned` is created an initialized to the remainder `power10%13 = 9` and we enter the for-loop where these things happen repeatedly:
`remainder=9*10=90, 90/13=6` is pushed onto `v={7,6}` and `remainder = 90%13=12`.
`remainder = 120, 120/13=9` is pushed, `v={7,6,9}`, `remainder = 120%13 = 3`,
`remainder = 30, 30/13 = 2` is pushed, `v={7,6,9,2}`, `remainder = 30%13 = 4`
`remainder = 40, 40/13 = 3` is pushed, `v={7,6,9,2,3}`, `remainder = 40%13 = 1`
`remainder = 10, 10/13 = 0` is pushed, `v={7,6,9,2,3,0}`, `remainder = 10%13 = 10`
And then the sequence of 6 values repeats over and over until `v` contains 101 values.

(b) On an attached page, write a program to define `findRepetend()`. Use the flow chart at right, it it helps. If the user enters 13, the output should read "`13 has a repetend of length 6`". Remember to use proper syntax and style and to define your variables.




```
unsigned findRepetend(vector<short> decimalDigits) {
    unsigned n{1}, i{0};
    while(n<50) {
        while(decimalDigits[i]==decimalDigits[i+n]) ++i;
        if(i>49) return n;
        else {
            ++n;
            i=0;
        }
    }
}
```