CS 31 Introduction to CS 1 Discussion 2H

Some Pointers on Project 1

- Organize your reports properly! Don’t write the entire hw into 1 paragraph
  - Use Bullets!
- Specific examples of what errors you introduced will be better instead of “explaining” the error:
  - The error that I introduced to create a logic error was changing the order of operations in the calculations of the pctNewson and pctCox.
  - 100/a*total instead of 100*a/total

Input	Error
Total # of voters: 9999999999	The program skipped the questions asking how many voted for Newsom and Cox and printed that 0.0% will vote for Newsom and -12.7% for Cox. This error occurred for all inputs of more than 9 integers.
Total # of voters: 100, Newsom #: 377, Cox #: 299	The program computed percents that were mathematically correct but illogical since they were over 100%.
Total # of voters: 0, Newsom #: 0, Cox #: 0	The program calculated "nan%" for both Newsom and Cox because it tried to divide 0 by 0. It predicted that Cox will win because the "else" statement executes since there is no check for equivalent numbers of votes for Cox and Newsom.
Total # of voters: 0, Newsom #: 99, Cox #: 80	The program calculated "inf%" for both Newsom and Cox because it divided an integer by 0.
Total # of voters: -36, Newsom #: -111, Cox #: -7	The program's computations are still mathematically correct with negative integers even though they make no sense in context of the question. It calculates 308.3% for Newsom and 19.4% for Cox and predicts that Cox will win because it compares values of the voter counts, not the percentages.

Logic error:
I changed lines 20 & 21 to:
double pctNewsom = 100.0 * numberSurveyed / forNewsom;
double pctCox = 100.0 * numberSurveyed / forCox;
The program now divides in the wrong order. For example when the input is total # of voters = 100, Newsom voters = 60, Cox voters = 40, the output is 166.7% for Newsom and 250.0% for Cox. The program still correctly predicts a Newsom victory since its comparing counts, not percents.

Compilation error:
I removed the line "**using namespace** std;" and received error messages because "cout" and "cin" were no longer defined.
I also removed line 11 **"int forCox;"** and the build failed because now when "forCox" appears elsewhere in the program, it's an undeclared identifier.

Variables, Control Flow, Loops

Variables

- Variables
  - A portion of memory to store data / a value.
  - Must be declared before first use
- Types
  - int (4 byte): integer
  - double (8 byte): real numbers
  - bool (1 byte): boolean, true or false
  - char (1 byte): character
  - string (varies): revisit later

Operators

- Arithmetic operators: +, -, *, /, %
  - % (mode): compute the remainder

- Shorthand operators
  - Increment (postfix & prefix):
    - i++; ++i;
  - Decrement (postfix & prefix):
    - i--; --i;
  - Compound assignments:
    - +=, -=. *=, /=. %=

Example	Equivalent To
`x += 2;`	`x = x + 2;`
`x -= 2;`	`x = x - 2;`
`x *= 2;`	`x = x * 2;`
`x /= 2;`	`x = x / 2;`
`x %= 2;`	`x = x % 2;`
`x *= a + b;`	`x = x * (a + b);`
`x++;`	`x += 1; x = x + 1;`
`x--;`	`x -= 1; x = x - 1;`

- Logical operators: && (AND), ll (OR); ! (NOT)
- Relational operators: ==, !=, >=, <=, >, <

Precedence Rules

- If not sure, use () to include expressions to control the order.
- Evaluate operator with higher precedence (with smaller number in the list on the next slide) first.
- Example:
  - `a || b && c`
- Source: http://en.cppreference.com/w/cpp/language/operator_precedence

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right
2	`a++ a--`	Suffix/postfix increment and decrement	
	type`()` *type*`{}`	Functional cast	
	`a()`	Function call	
	`a[]`	Subscript	
	`. ->`	Member access	
3	`++a --a`	Prefix increment and decrement	Right-to-left
	`+a -a`	Unary plus and minus	
	`! ~`	Logical NOT and bitwise NOT	
	(*type*)	C-style cast	
	`*a`	Indirection (dereference)	
	`&a`	Address-of	
	`sizeof`	Size-of[note 1]	
	`new new[]`	Dynamic memory allocation	
	`delete delete[]`	Dynamic memory deallocation	
4	`.* ->*`	Pointer-to-member	Left-to-right
5	`a*b a/b a%b`	Multiplication, division, and remainder	
6	`a+b a-b`	Addition and subtraction	
7	`<< >>`	Bitwise left shift and right shift	
8	`< <=`	For relational operators < and ≤ respectively	
	`> >=`	For relational operators > and ≥ respectively	
9	`== !=`	For relational operators = and ≠ respectively	
10	`a&b`	Bitwise AND	
11	`^`	Bitwise XOR (exclusive or)	
12	`|`	Bitwise OR (inclusive or)	
13	`&&`	Logical AND	
14	`||`	Logical OR	

cin & getline

- cin:
  - Only reads until the first whitespace (break), Except some leading whitespace.
- getline(cin, str):
  - Stores the entire line of input into str.
  - Does not break on whitespace.

- The major difference: getline allows the user to type in a string, store the whole line typed in, and will not break upon whitespace.

Control Flow

- Sometimes, we want to make our programs behave differently based on different conditions.
- if statements can be used to archive conditions.
- Example:

```
if(tommorow rains){
    I will not go to school.
}
else{
    I will go to school.
}
```

if statement

- Syntax:

```
if (expression) {
    statement(s);
}
```

  - If a single statement inside, braces can be omitted, although you'd better keep them, in case that you may want to add more statement(s) afterwards.
- Rules:
  - If (and only if) the expression is evaluated to true (non-zero value): execute the statement(s) inside.
  - If the expression is false: the statement(s) will not be executed (i.e. simply ignored)
  - Then the program continues right after the entire if statement.

if statement

- Practice: will it be printed?

```
int main() {
    int x = 100;
    if(x == 100){
        cout << "x is 100";
    }
}
```

```
int main() {
    int x = 10;
    if(x == 100){
        cout << "x is 100";
    }
}
```

if statement

- Compare: are they the same?

```
if(x == 100){
    cout << "x is 100";
}
```

```
if(x = 100){
    cout << "x is 100";
}
```

- Common pitfall: “==” is not the same as “=”
  - “==”: relational operator, to check if the value of left-hand side equal to the value of right-hand side
  - “=”: assignment operator, to assign the value of right-hand side expression to the variable on the left-hand side.

if.. else statement

- Syntax:

```
if (expression) {
    statement(s)_1;
}
else {
    statement(s)_2;
}
```

  - If a single statement inside, braces can be omitted.
- Rules:
  - If the expression is true: execute statement(s)_1
  - If the expression is false: ignore statement(s)_1, execute statement(s)_2
  - The program continues right after the entire if.. else statement.
- Example:

```
if (x == 100)
    cout << "x is 100";
else
    cout << "x is not 100";
```

if.. else if.. else statement

- Syntax and example:

```
if (expression_1) {
    statement(s)_1;
}
else if (expression_2) {
    statement(s)_2
}
else if (expression_3) {
    statement(s)_3
}
else {
    statement(s)_4;
}
```

```
if (x > 0) {
    cout << "x is positive";
}
else if (x < 0) {
    cout << "x is negative";
}
else {
    cout << "x is zero";
}
```

- Multiple if + else structures can be concatenated to check a range of values
- There can be several "`else if (condition) { statement(s) }`" in the structure.
- In this structure, the conditions / ranges are exclusive to each other.

Nested if statement

- Put another if statement inside an if statement
- Can be used to check layered conditions
  - eg. to shrink the size of proper candidates from several rounds of check.
- Example:

```
int main() {
    int x, y;
    x = y = 10;

    if(x == 10)
        if (x == y)
            x++;

    if(x != y)
        x++;
    cout << x << endl;
}
```

```
int main() {
    int x = 9, y = 10, z = 0;

    if(x == 9)
        if (x == y)
            x++;

    else
        z = 1;
    cout << z << endl;
}
```

- Dangling else:
  - else statement will pair with the last unmatched if statement regardless the indentation (although it helps to make your code readable.

Common Pitfalls

- Output?

```
int main() {
    int x = 7;
    if(x == 7)
        int y = 8;
    cout << y << endl;
}
```

```
int main() {
    int x = 1;
    int y;
    if(x == 1){
        y = x;
        int z = y * y;
        y = z + 2;
    }
    cout << y << endl;
}
```

- Compile error:
  - y is not declared in the proper scope.
  - if y is declared within the if statement, it cannot be seen outside the if statement. (more on scope next time.)
- Note: if you have multiple statements, you have to use braces to include them (don't need to do so if it's a single statement like shown left).

Loops

- Loops repeat statement(s) a certain number of times, or while a condition is evaluated to true.
- Keywords: while, do, for

The for Loop

- Syntax:

```
for (initialization; stay_in_loop_condition; prepare_for_next_iteration) {
    statement(s)
    };
```

- for loops repeat statement(s) while stay-in-loop condition is **true (non-zero)**.
- Rule:
  - Initialization: execute before entering the loop
    - Do NOT do initialization in the statement of for loops — infinite loop error
  - Stay-In-Loop-Condition:
    - If **true (non-zero)**, enter the loop and execute the statement;
    - If **false**, exit the loop, and continues the rest of the whole program
  - Prepare-For-Next-Iteration: execute after the statement executed
    - In many cases, this will be some increment / decrement such as i++, --j;

The while Loop

- Syntax:

```
while (expression) {
    statement(s)
}
```

- Rules:
  - The while-loop simply **repeat**s statement(s) **while** the expression is **true** (non-zero value).
  - If the condition is no longer true after any execution of statement(s):
    - The loop ends
    - And the program continues right after the loop
- Compare:

```
int n = 10;

while (n>0) {
    cout << n << ", ";
    --n;
}

cout << "The End!\n";
```

```
while (n>0) {
    int n = 10;

    cout << n << ", ";
    --n;
}

cout << "The End!\n";
```

The while Loop

- Syntax:

```
while (expression) {
    statement(s)
}
```

- Rules:
  - The while-loop simply **repeat**s statement(s) **while** the expression is **true** (non-zero value).
  - If the condition is no longer true after any execution of statement(s):
    - The loop ends
    - And the program continues right after the loop
- Compare:

```
int n = 10;

while (n>0) {
    cout << n << ", ";
    --n;
}

cout << "The End!\n";
```

```
while (n>0) {
    int n = 10;

    cout << n << ", ";
    --n;
}

cout << "The End!\n";
```

Avoid infinite loop.

The do-while Loop

- Syntax:

```
do {
    statement(s)
} while (expression);
```

Never miss the semicolon here!!

- Rules:
  - Act like a while loop
  - **Except** that the expression is evaluated after one execution of the statement (instead of checking condition before statements execution). It guarantees at least one execution of the statements, even if the condition is never fulfilled.
- Difference between while and do-while loop:
  - The while-loop: Check condition first, then execute.
  - The do-while-loop: Execute first, then check condition. So it execute at least once.

The do-while Loop

- Preferred when the statement need to be executed at least once.

```
string str;
do {
    cout << "Enter text: ";
    getline (cin,str);
    cout << "You entered: " << str << '\n';
} while (str != "goodbye");
```

- while loop VS do-while loop

```
int x = 0;
while (x < 0) {
    cout << x << endl;
    x++;
}
```

```
int x = 0;
do {
    cout << x << endl;
    x++;
} while (x < 0);
```

The do-while Loop

- Preferred when the statement need to be executed at least once.

```
string str;
do {
    cout << "Enter text: ";
    getline (cin,str);
    cout << "You entered: " << str << '\n';
} while (str != "goodbye");
```

- while loop VS do-while loop

```
int x = 0;
while (x < 0) {
    cout << x << endl;
    x++;
}
```

nothing printed

```
int x = 0;
do {
    cout << x << endl;
    x++;
} while (x < 0);
```

0 printed

for VS while Loop

- They can be used equivalently:

```
for (init; cond; next) {
        statement;
}
```

```
init;
while (cond;) {
        statement;
        next;
}
```

Nested Loop

- Nested: place one loop inside the body of another loop
- All types of loops can be nested
- Different types of loops can be nested
- **The outer loop move one step forward only after the inner loop is completely finished**
- Example:
  - What is the output? Can you write it in a different way?

```
for(int i = 0; i < 10; i++) {
    for(int j = 0; j < 10; j++) {
        cout << i << j << " ";
    }
}
cout << endl;
```

Time for practicing the Worksheet

1. Circle where the bug occurs and explain what incorrect behavior will happen. What do you think this program will output? Add a fix.

```
cout << "Enter your name: ";
getline( cin , name );

cout << "\nEnter your UID: ";
int UID;
cin >> UID ;

cout << "\nEnter your Major: ";
getline( cin , major );

cout << "\nEnter your residence hall: ";
getline( cin , hall );
```

1. Circle where the bug occurs and explain what incorrect behavior will happen. What do you think this program will output? Add a fix.

```
cout << "Enter your name: ";
getline( cin , name );

cout << "\nEnter your UID: ";
int UID;
cin >> UID ;

cout << "\nEnter your Major: ";
getline( cin , major );

cout << "\nEnter your residence hall: ";
getline( cin , hall );
```

Bug: It will skip "Enter your Major", because getline has already consumed a newline character.

A newline is always appended to your input when you select Enter or Return when submitting from a terminal. It is also used in files for moving toward the next line. When the flow of control reaches std::getline(), the newline will be discarded, but the input will cease immediately. The reason this happens is because the default functionality of this function dictates that it should (it attempts to read a line and stops when it finds a newline).

Because this leading newline inhibits the expected functionality of your program, it follows that it must be skipped our ignored somehow. One option is to call cin.ignore(10000, '\n') after the the first extraction. It will discard the next available character so that the newline is no longer intrusive.

1. What is the output of the following code?

```
int a = 10;
int b = 22;
while (a / 2 >= 1) {
  a--;
  cout << a << endl;
  if ((a + b) % 2 == 0) {
    a--;
    cout << a << endl;
    b /= 2;
  }
}
```

1. What is the output of the following code?

```
int a = 10;
int b = 22;
while (a / 2 >= 1) {
  a--;
  cout << a << endl;
  if ((a + b) % 2 == 0) {
    a--;
    cout << a << endl;
    b /= 2;
  }
}
```

9
8
7
6
5
4
3
2
1

3. This code snippet tries to print all prime numbers between 3 and a given input *n*. Find the 3 bugs contained in the code and fix them.

```
int n;
cin >> n;
for (int candidate = 3; candidate < n; ++candidate) {
      bool isPrime = true;
      for (int x = 2; x < n; x++) {
            if (candidate % x = 0){
                  isPrime = false;
            }
      }

      if (isPrime) {
            cout << n << " ";
      }
}
```

```
int n;
cin >> n;
int candidate = 3
while (candidate < n) {
        bool isPrime = true;
        for (int x = 2; x < candidate; x++) {
                if (candidate % x == 0){
                        isPrime = false;
                }
            }

            if (isPrime) {
                    cout << candidate << " ";
            }
            candidate = candidate + 1;
    }
```

Programming Problems

1. Write a program that takes in a number as an int and outputs the sum of all of the digits in that number

Sample Output:
Enter a number: 184
The sum of the digits in your number is 13!

main.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
  int n;
    cout<<"Enter a number";
    cin>>n;
    //Find length of the number
    int no = n, digits=0;
    while(no>0)
    {
      digits+=1;
      no/=10;
    }
    cout<<"Number of digits in the number "<<digits<<endl;
  //Sum over all digits
  int sum=0;
  no = n;
  while(no>0)
  {
    sum+=(no%10);
    no/=10;
  }
  cout<<"Sum of digits in the number "<<sum;
}
```

2. Write a program that takes in N numbers outputs the average of the N numbers.

Sample output:
How many numbers do you want to average? 5
Number: 4
Number: 2
Number: 8
Number: 9
Number: 7
The average is 6

main.cpp

```cpp
#include <iostream>
using namespace std;
int main() {

  int n, i=0, no;

  cin>>n;
  double sum=0;
  for(i=0;i<n;i++)
  {
    cin>>no;
    sum+=no;
  }
  cout<<"avg "<<sum/n;
}
```

3. Write a program that takes in N integers and outputs the sum of the even numbers.

Sample output:
How many numbers are you entering? 4
Enter a number: 2
Enter a number: 9
Enter a number: -3
Enter a number: 6
Result: 8

```cpp
#include <iostream>
using namespace std;
int main() {
  int n, i=0, no, sum=0;
  cout<<"How many numbers are you entering ?";
  cin >>n;
  for(i=0;i<n;i++)
  {
    cin>>no;
    if (no%2==0) //check if the number is even
    sum+=no;
  }
  cout<<"Result "<<sum;
}
```

4. Write a program that takes in two numbers and a command of type string (“Add”, “Subtract”, “Multiply”, “Divide”). Inputting an invalid command should cause the program to ask for a valid command.

Sample output:
Enter your first number: 3
Enter your second number: 7
Enter your command: Multiply
Result: 21

```
int main() {
    int first = 0;
    int second = 0;
    string command = "";
    cout << "Enter your first number: ";
    cin >> first;
    cout << "Enter your second number: ";
    cin >> second;
    cin.ignore(10000, '\n');

    cout << "Enter your command: ";
    getline(cin, command);
    if (command == "Add")
        cout << "Result: " << first + second << endl;
    else if (command == "Subtract")
        cout << "Result: " << first - second << endl;
    else if (command == "Multiply")
        cout << "Result: " << first * second << endl;
    else if (command == "Divide" && second != 0)
        cout << "Result: " << first / second << endl;
    else {
        cout << "Invalid command!" << endl;
    }
}
```

5. Write a program that reads in an integer N and prints an NxN box where the (i,j)th character is as follows:

```
'.'   if j > i
i + j  otherwise
```

Where i is the row number and j is the column number (starting at 0, not 1). For Example, if the input is 4, it should print:

```
0 . . .
1 2 . .
2 3 4 .
3 4 5 6
```

main.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
  int n, i, j;
  cin>>n;
  for(i=0; i<n; i++)
  {
    for(j=0; j<n; j++)
    {
      if(j>i)
        cout<<". ";
      else
        cout<<i+j<<" ";
    }
    cout<<endl;
  }
}
```

7. Write a program that takes in an integer N where N > 0, and outputs a comma-separated list of all the factors of N.

Sample input:
12
Sample output:
1,2,3,4,6,12

```
int main() {
        int n;
        cin >> n;

        cout << "1";      // 1 is always a factor
        for (int i = 2; i <= n; i++) {
                if (n % i == 0) {
                        cout << "," << i;
                }
        }
        cout << endl;
        return 0;
}
```

6. Write a program that reads in an integer and prints whether that number is a perfect number. A perfect number is defined as a number that is equal to the sum of all factors excluding itself.

Example:

4 != 1 + 2 => Print “Not perfect.”

5 != 1 => Print “Not perfect.”

6 = 1 + 2 + 3 => Print “Perfect.”

12 != 1 + 2 + 3 + 4 + 6 => Print “Not perfect.”

28 = 1 + 2 + 4 + 7 + 14 => Print “Perfect.”

main.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
  int n, i, sum=0;
  cin>>n;
  for(i=1;i<n;i++)
  {
    if (n%i==0)
    {
      sum+=i;
    }
  }
  if(sum==i)
    cout<<"Perfect";
  else
    cout<<"Not Perfect";
}
```

8. Write a program that given an input integer N, finds an integer x such that $2^x <= N < 2^{x+1}$. The program should ask for user input and print the integer x it finds. If there exists no such x, it should print “error”.

Sample Input:

200 => Should output 7, since $2^7 = 128 <= 200 < 2^8 = 256$.

20 => Should output 4, since $2^4 = 16 <= 20 < 2^5 = 32$.

main.cpp

```cpp
#include <iostream>
using namespace std;
int main() {
  int n, i, x=0, prod=2;
  cin>>n;
  while(prod<=n)
  {
    prod*=2;
    x+=1;
  }
  if(n<=0)
    cout<<"Error";
  else
    cout<<x;
}
```