CS31: Discussion 1G - Week 2

Notes about project 1

- Name your report files correctly.
- Common wrong file names - report.txt.txt report.docx.txt
- Read requirements clearly - some students have given just 1 compile error when the assignment required 2
- Read comments even if you got 100

Office Hours

- Thursday- 09.30-11.30 - Boelter 3256s
- Feel free to attend office hours of any TA.

How to verify file-names you are submitting are accurate?

After you have copied the zip file to seasnet to verify and unzipped the zip file into a directory as per instructions here. You can run the `ls` command to list the files in the directory.

`ls`

The output of the command should be same as all the files you have to submit as per the project spec. So for Project-1 the output should have been
`compile_error.cpp logic_error.cpp original.cpp report.txt`

or `compile_error.cpp logic_error.cpp original.cpp report.docx`

Overview

- String handling detail - cin, getline
- Type compatibility and conversion
- Conditional Expressions
- Selective control
- Looping constructs

String Handling

Recap of cin and getline

```
//learn about cin

#include <iostream>
#include <string>
using namespace std;

int main()
{
    string a = "this is a test";
    string b = "this ";
    string c = "this";
    string input;
    cout << "Enter your input" << endl;
    //type in
    //"this is a test"
    // and then hit return/enter
    cin >> input ;
    cout << "Is input equal to a? "<< (input == a) << endl;
    cout << "Is input equal to b? " << (input == b) << endl;
    cout << "Is input equal to c? " << (input == c) << endl;
    return(0);
}
```

```
//learn about getline

#include <iostream>
#include <string>
using namespace std;

int main()
{
    string a = "this is a test\n";
    string b = "this is a test ";
    string c = "this is a test";


    string input;

    cout << "Enter your input" << endl;
    //type in
    //"this is a test"
    // and then hit return/enter
    getline(cin,input);

    cout << "Is input equal to a? "<< (input == a) << endl;
    cout << "Is input equal to b? " << (input == b) << endl;
    cout << "Is input equal to c? " << (input == c) << endl;

    return(0);
}
```

`getline( cin , aString )` reads a texual line ending with `\n`, consuming the `\n` character itself, which means that `\n` is not part of the input string.

cin.ignore

```
//learn about cin.ignore

#include <iostream>
#include <string>
using namespace std;

int main()
{
    string input;

    cout << "enter your input " << endl;
    cin.ignore(5, '\n');
    //type in 123456789
    cin >> input ;

    cout << "input read was " <<input << endl;

    return 0;
}
```

`cin.ignore( 1000, ‘\n’)` discards the next \n character or 1000 characters, which ever comes first.

Why do we need cin.ignore?

Consider the below program.

```
//learn about cin.ignore

#include <iostream>
#include <string>
using namespace std;

int main()
{
    int numberInput;
    string stringInput;

    cout << "enter numberInput ";
    cin >> numberInput ;

    cout << "enter stringInput ";
    getline(cin,stringInput);

    cout << "\n";
    cout << "numberInput was " << numberInput << endl;
    cout << "stringInput was " << stringInput << endl;

    return 0;
}
```

When you run this program, you will notice that before you can enter the input for stringInput, the program exits. This is because when you hit `\n` after entering the numberInput, `getLine` reads the `\n` and breaks.

How to use cin.ignore to fix this?

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int numberInput;
    string stringInput;

    cout << "enter numberInput ";
    cin >> numberInput ;
    cin.ignore(10000,'\n');

    cout << "enter stringInput ";
    getline(cin,stringInput);

    cout << "\n";
    cout << "numberInput was " << numberInput << endl;
    cout << "stringInput was " << stringInput << endl;

    return 0;
}
```

cin.ignore declaration position.

A question was asked in class, does it matter where the cin.ignore is declared. The answer is yes, it does.

Consider the below program:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int numberInput1, numberInput2;
    string stringInput1, stringInput2;

    cout << "enter numberInput1 ";
    cin >> numberInput1 ;


    cout << "enter stringInput1 ";
    getline(cin,stringInput1);

    cout << "enter numberInput2 ";
    cin >> numberInput2 ;

    cin.ignore(10000,'\n');

    cout << "enter stringInput2 ";
    getline(cin,stringInput2);


    cout << "\n";
    cout << "numberInput1 was " << numberInput1 << endl;
    cout << "stringInput1 was " << stringInput1 << endl;
    cout << "numberInput2 was " << numberInput2 << endl;
    cout << "stringInput2 was " << stringInput2 << endl;

    return 0;
}
```

The input output for the above program looks like

```
enter numberInput1 10
enter stringInput1 enter numberInput2 11
enter stringInput2 abcdefghi jklmnopqr

numberInput1 was 10
stringInput1 was
numberInput2 was 11
stringInput2 was abcdefghi jklmnopqr
Program ended with exit code: 0
```

Notice that I wasn't able to type in an input for stringInput1, because the `cin.ignore` was declared only after numberInput2. So the inputstream characters were ignored only after the declaration, which happened after numberInput2 was given.

Type compatibility and conversion - courtesy Absolute CPP textbook

Variables of one data-type can be converted to other data-type. This is called type casting.

Simple example:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    cout << 9/2 << endl;
}
```

making one of the operands as a floating point makes it a floating point division.

Ex:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    cout << 9/2.0 << endl;
    cout << 9.0/2 << endl;
}
```

What if the 9 and the 2 are the values of variables of type int named n and m?

If you want floating-point division in this case, you must do a type cast from int to double (or another floating-point type), such as in the following:

`double ans = n/static_cast<double>(m);`

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int n = 9, m = 2;
    double ans = n/static_cast<double>(m);
    cout << ans << endl;
}
```

The expression `static_cast<double>(m)` is a type cast. The expression static_cast is like a function that takes an int argument (actually, an argument of almost any type) and returns an "equivalent" value of type double. So, if the value of m is 2, the expression `static_cast<double>(m)` returns the double value 2.0.

static_cast(n) does not change the value of the variable n. If n has the value 2 before this expression is evaluated, then n still has the value 2 after the expression is evaluated.

How about static cast to int?

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    double dbl = 1.234;
    int i =static_cast<int>(dbl);
    cout << dbl << endl;
    cout << i << endl;
}
```

Notice that after converting from double to int, we are losing precision. So this is not recommended.

Type coercion

We can assign a value of an integer type to a variable of a floating-point type, as in

```
double d = 5;
```

in this case C++ does the type-cast for us. Such an automatic conversion is sometimes called a type coercion.

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int a = 5;
    double b = a;

    cout << "a is " << a << endl;
    cout << "b is " << b << endl;

    return 0;
}
```

When working with `A operand B` where operand may be +, -, *, /, or %

if A or B is double, the result will be double

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    cout << 3 + 4.4 << endl;
    cout << 2.2 * 3 << endl;
    cout << 2.2 * 3.0 << endl;
    cout << 2 * 3 << endl;
    cout << 2 * 3.0 << endl;
    cout << 4.5 * 2 << endl;
    return 0;
}
```

Common Pitfall: Integer Division

Example:

- 7 / 3 evaluates to 2
- 7.0 / 3 evaluates to 2.333…
- 7 / 3.0 evaluates to 2.333…
- 7.0 / 3.0 evaluates to 2.333…

```
#include <iostream>
using namespace std;

int main()
{
    int i = 17 / 2 + 4;
    double d1 = 17 / 2 + 4;
    double d2 = 17 / 2.0 + 4;
    double d3 = 17 / 2 + 4.0;

    cout << i << endl;
    cout << d1 << endl;
    cout << d2 << endl;
    cout << d3 << endl;
    return 0;
}
```

Want more problems?: Look at Worksheet 1 problem 3

Conditional Expressions

Expressions that evaluate to (true or false) 0 or 1. For example:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    cout << "2==2? " << (2==2) << endl;
    cout << "3==3? " << (3==3) << endl;
    cout << "3==2? " << (3==2) << endl;
    cout << "2==3? " << (2==2) << endl;

    return 0;
}
```

More examples; taken from worksheet 1:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int x,y,z,t;
    x=-2;
    y=5;
    z=0;
    t=-4;

    cout << (x + y < z + 1) << endl;
    cout << (x - 2 * y + y < z * 2 / 3) << endl;
    cout << (3 * y / 4 < 8 && y >= 4) << endl;
    cout << (t > 5 || z < 2) << endl;
    cout << (x * y < 10 || y * z < 10) << endl;

    return 0;
}
```

&& operator

The && operator evaluates to 0 or 1 based on the operation being performed.

If both conditions on the left and right evaluate to true, && evaluates to true.

ex:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
   int x,y,z,t;
   x=0;
   y=1;
   z=2;
   t=3;

    int a = (x < y )&& (z < 3);
    int b = (x > y )&& (z < 3);

    cout << "a value " << a << endl;
    cout << "b value " << b << endl;

    return 0;
}
```

What will below program output?

```
#include <iostream>
#include <string>
using namespace std;

int main()
{

    int a = 1 && 10;
    int b = 7 && 1000;
    int c = 0 && 12;

    cout << "a value " << a << endl;
    cout << "b value " << b << endl;
    cout << "c value " << c << endl;

    return 0;
}
```

&& checks if both left and right are non-zero values.

|| operators

The || operator evaluates to 0 or 1 based on the operation being performed.

If **any one** of the conditions on the left or right evaluate to true, || evaluates to true.

ex:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
   int x,y,z,t;
   x=0;
   y=1;
   z=2;
   t=3;

    int a = (x < y )|| (z < 3);
    int b = (x > y )|| (z < 3);

    cout << "a value " << a << endl;
    cout << "b value " << b << endl;

    return 0;
}
```

```
#include <iostream>
#include <string>
using namespace std;

int main()
{

    int a = 1 || 10;
    int b = 7 || 1000;
    int c = 0 || 12;

    cout << "a value " << a << endl;
    cout << "b value " << b << endl;
    cout << "c value " << c << endl;

    return 0;
}
```

Short circuit evaluation

What does this mean? The evaluation of the expression stops

- once the condition turns false (in case of &&)
- true (in case of ||)

Key points

- Learn precedence rules.
- Use parenthesis generously.
- Check if the operands have decimal points.
- Take care of integer division

Selective control

if-else statements

Ex: Print a specific message for Dwight based on the day.

Dwight goes farming only on sunday.

```

#include <iostream>
#include <string>
using namespace std;

int main()
{

    string day = "Sunday";

    if(day == "Sunday")
        cout << "Today is Sunday, Dwight Schrute is going to his Beet farm!";

    return 0;
}
```

Print a different message if day is something other than sunday.

```
#include <iostream>
#include <string>
using namespace std;

int main()
{

    string day = "Monday";

    if(day == "Sunday")
        cout << "Today is Sunday, Dwight Schrute is going to his Beet farm!" << endl;
    else
        cout << "Today is not Sunday, Dwight Schrute is going to his office - Dunder
Mifflin" << endl;

    return 0;
}
```

Change above program so that Dwight goes to the mall on Saturdays, beet farm on sunday and office on weekdays.

```
#include <iostream>
#include <string>
using namespace std;

int main()
{

    string day = "Saturday";

    if(day == "Sunday")
        cout << "Today is Sunday, Dwight Schrute is going to his Beet farm!" << endl;
    else if(day == "Saturday")
        cout << "Today is a Saturday, Dwight Schrute is going to the mall." << endl;
    else
        cout << "Today is a weekday, Dwight Schrute is going to his office - Dunder M
ifflin" << endl;

    return 0;
}
```

We just learnt if, else-if and else statements.

I want to print two statements for each of the above conditions

will below program work?

```
#include <iostream>
#include <string>
using namespace std;

int main()
{

    string day = "Saturday";

    if(day == "Sunday")
        cout << "Today is " << day << endl;
        cout << "Today is Sunday, Dwight Schrute is going to his Beet farm!" << endl;

    else if(day == "Saturday")
        cout << "Today is " << day << endl;
        cout << "Today is a Saturday, Dwight Schrute is going to the mall." << endl;
    else
        cout << "Today is " << day << endl;
        cout << "Today is a weekday, Dwight Schrute is going to his office - Dunder M
ifflin" << endl;

    return 0;
}
```

No. We need to parenthesize as below.

```
#include <iostream>
#include <string>
using namespace std;

int main()
{

    string day = "Saturday";

    if(day == "Sunday"){
        cout << "Today is " << day << endl;
        cout << "Today is Sunday, Dwight Schrute is going to his Beet farm!" << endl;
    }

    else if(day == "Saturday")
    {
        cout << "Today is " << day << endl;
        cout << "Today is a Saturday, Dwight Schrute is going to the mall." << endl;
    }

    else
    {
        cout << "Today is " << day << endl;
        cout << "Today is a weekday, Dwight Schrute is going to his office - Dunder M
ifflin" << endl;
    }

    return 0;
}
```

Nested if-else

```
#include <iostream>
#include <string>
using namespace std;

int main()
{

    string day = "Saturday";

    if(day == "Sunday"){
        cout << "Today is " << day << endl;
        cout << "Today is Sunday, Dwight Schrute is going to his Beet farm!" << endl;
    }

    else if(day == "Saturday")
    {
        cout << "Today is " << day << endl;
        cout << "Today is a Saturday, Dwight Schrute is going to the mall." << endl;
    }

    else
    {
        if(day == "Monday")
        {
            cout << "Mondays are awesome!!" << endl;
        } else if(day == "Friday"){
            cout << "Thank god its Friday!!" << endl;
        }

        cout << "Today is " << day << endl;
        cout << "Today is a weekday, Dwight Schrute is going to his office - Dunder M
ifflin" << endl;
    }

    return 0;
}
```

can we have else or else-if on their own, without an if-statement.

No.Both below programs are invalid

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string day = "Sunday";
      else
        cout << "Today is " << day << endl;
        cout << "Today is a weekday, Dwight Schrute is going to his office - Dunder M
ifflin" << endl;

    return 0;
}
```

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string day = "Sunday";
    else if(day="Sunday")
        cout << "Today is " << day << endl;
    cout << "Today is a weekday, Dwight Schrute is going to his office - Dunder Miffl
in" << endl;

    return 0;
}
```

Switch statements

When we have too many if else-if and else statements it gets confusing.

Example:

Print day of the week given a number from 1-7.

1 - monday 2 - tuesday 3 - wednesday 4 - thursday ... 7 - sunday

```
#include <iostream>
#include <string>
using namespace std;

int main()
{

    int dayNum = 1;

    switch (dayNum) {
        case 1:
            cout << "Monday" << endl;
            break;
        case 2:
            cout << "Tuesday" << endl;
            break;
        case 3:
            cout << "Wednesday" << endl;
            break;
        case 4:
            cout << "Thursday" << endl;
            break;
        case 5:
            cout << "Friday" << endl;
            break;
        case 6:
            cout << "Saturday" << endl;
            break;
        case 7:
            cout << "Sunday" << endl;
            break;
        default:
            cout << "Number has to be in 1-7" << endl;
            break;
    }
    return 0;
}
```

Common Gotcha's

- Dont forget the break; after every case -> logic-error
- Dont have two cases with same value -> compile-error

output of below program? ```

include

include

using namespace std;

int main() {

```
int dayNum = 1;
```

// dayNum = 2; // dayNum = 3; // dayNum = 6;

```
switch (dayNum) {
    case 1:
        cout << "Monday" << endl;
    case 2:
        cout << "Tuesday" << endl;
    case 3:
        cout << "Wednesday" << endl;
        break;
    case 4:
        cout << "Thursday" << endl;
        break;
    case 5:
        cout << "Friday" << endl;
        break;
    case 6:
        cout << "Saturday" << endl;
        break;
    case 7:
        cout << "Sunday" << endl;
        break;
    default:
        cout << "Number has to be in 1-7" << endl;
        break;
}
return 0;
```

}

```
## Looping constructs

Looping constructs like for, while and do-while are provided for us to execute certai
n instructions repeatedly.

Ex:

* Print the word "Go Bruins!" ten times.
* Add the number 1 ten times
* Increase a number by 1 and print till its less than another number

### while loops
Repeat while a condition is true.
```

while(logical expression){ //…execute a block of statements; } ```

ex:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int x = 1, y = 10;
    while(x<y){
        cout<<x<<endl;
        x = x+1;
    }
}
```

do while loops

```
do{
//…execute a block of statements;
}while(logical expression);
```

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int x = 1, y = 10;
    do{
        cout<<x<<endl;
        x = x+1;
    }while(x<y);
}
```

what happens to below program

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int x = 10, y = 10;
    do{
        cout<<x<<endl;
        x = x+1;
    }while(x<y);
}
```

still outputs the answer, as check happens **after** 1 execution of the block of statements.

For loop

As can be seen in above two looping constructs, both change a certain value, make a certain check and execute a block of code repeatedly.

For loop provides direct syntax to do perform these actions.

```
for(i = 1; i<n; i++)
{
//block of statements
}
```

execution order:

1. test the condition
2. if it is false, terminate
3. if it is true, execute the body
4. execute the incrementation step

Ex:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int i;

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

Exercise:

1. Using while, do-while and for loop, write a program to print numbers from 100 to 1. i.e., the sequence should be like

```
100
99
98
...
2
1
```

1. Change the above program so that only odd numbers are printed out

```
99
97
...
3
1
```

1. Write a program to print the multiplication tables for a number. The number should be input from a user.

Ex: output for user input of 2 should be

```
2 x 1 = 2
2 x 2 = 4
...
...
...
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```