CSI 140 Introduction to Programming

Converting from while(true) to while loop

A while(true) loop runs forever until it hits a break statement. A regular while loop is often clearer because the exit condition is visible right at the start of the loop, making the code easier to read and understand.

The Conversion Process

Step 1: Find the break condition Look inside your while(true) loop and find the if statement that contains break. This tells you when the loop should stop.

Step 2: Negate (flip) the condition The break condition tells you when to EXIT the loop. But the while condition needs to tell you when to CONTINUE the loop. So, you flip it to the opposite:

- **`>`** becomes **`<=`**
- **`<`** becomes **`>=`**
- **`==`** becomes **`!=`**
- **`true`** becomes **`false`**
- A boolean variable **`done`** becomes **`!done`**

Step 3: Move the negated condition to while Replace while(true) with while(negated_condition)

Step 4: Remove the if-break block Delete the entire if statement and break since they're no longer needed.

Example Walkthrough: Even Numbers from 2 to 20

Original with while(true):

```
int num = 2;
while (true) {
    cout << num << " ";
    num += 2;
    if (num > 20) {
        break;
    }
}
```

Conversion steps:

1. Break condition is: **`num > 20`**
2. Negate it: **`num <= 20`** (keep going while num is 20 or less)
3. Put in while: while (num <= 20)
4. Remove if-break block

Result:

```
int num = 2;
while (num <= 20) {
    cout << num << " ";
    num += 2;
}
```

Convert the following while(true) loop to a while loop and give the output:

```
int count = 10;
while (true) {
    cout << count << " ";
    count--;
    if (count < 1) {
        break;
    }
}
cout << "Blast off!" << endl;
```

output:

While loop:

```
int total = 0;
int num = 1;
while (true) {
    total += num;
    cout << "Added " << num << ",
total is now " << total << endl;
    num++;
    if (total >= 10) {
        break;
    }
}
```

output:

While loop:

Converting from while loop to for loop

A for loop is ideal when you know exactly how many times you want to iterate or when you're counting with a consistent pattern. It keeps all loop control (initialization, condition, update) in one clear line at the top.

The Conversion Process

Step 1: Identify the three components Look at your while loop and find:

- **Initialization**: Variable setup before the loop (e.g., int i = 0;)
- **Condition**: The while condition (e.g., i < 10)
- **Update**: How the variable changes each iteration (e.g., i++)

Step 2: Move them to the for loop header The for loop syntax is: for (initialization; condition; update)

Step 3: Remove redundant code Delete the initialization line before the loop and the update line inside the loop.

When to Use for vs while

Use for loop when:

- You know the exact number of iterations
- You're counting (incrementing/decrementing)
- Loop control is simple and fits in one line

Use while loop when:

- Number of iterations is unknown
- Condition is complex
- Multiple variables control the loop
- The loop depends on external events or user input

Example Walkthrough: Print Numbers 1 to 5

Original with while loop:	**Conversion steps:**

`int i = 1;    // Initialization`
`while (i <= 5) { // Condition`
`    cout << i << " ";`
`    i++;           // Update`
`}`
	1. Initialization: int i = 1
	2. Condition: i <= 5
	3. Update: i++

Result:

```
for (int i = 1; i <= 5; i++) {
    cout << i << " ";
}
```

2. Convert the following while-loop to a for-loop and a do-while loop.

```
int i = 1;
while (i < 10){
   cout << "Tag!\n";
   i++;
}
```

3. Convert the following for-loop to a do-while loop and a while loop.

```
int i;

for (i = 1; i <= 10; i += 3){
    cout << "i = " << i << endl;
}
```