CS 1436.009:

For Loops, When to Use Each Type of Loop,
Running Totals, Nested Loops, and
Breaking / Continuing.

Def. A for loop is a count-controlled control structure which iterates over a set no. of times.

Consider:

```
for (A; B; C) {
    // Code //
}
```

A: = iterator

B: = loop condition

C: = updating step

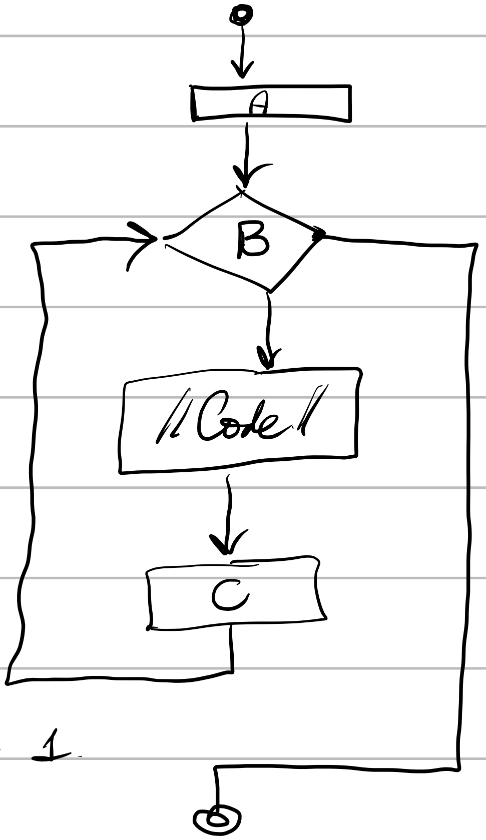


Figure 1.

Query 1. Are for loops pre/post-test?

if (condition) { 2 → };
⋮
if (condition) { 2 → };

~~for(A; B; C) { 2 → };~~

↑ do not do this! Bad style.

Query 2. for (A ; B; C) {
}

⇓

for(A0, A1 ; B; C) {
} ✓

⇒ for(int x=1, int y=2 ; x <= 4; x++)
{
std::cout << "sum is " << (x+y);
}

Query 3. for(int x=1, int y=2; x <= 4; x++, y++)
{
std::cout << "sum is " << (x+y);
}

A for loop can be modified further :-

```
→   int x=1;
    for(   ; B ; C ) {
    }
```

Query 4.

```
for ( "  ; B ;  ) {      ← header
    C
}
```

Running Totals

Def. a Running Total is that sum of numbers accumulated over the iterations of a loop.

Programming Example 1.

```
int num , sum =0 ;

for( num =1 ; num <= 10 ; num ++)
{
        sum += num;
}


std:: cout << sum << std::endl;
```

Programming Example 2

```
double dailySales, weeklySales =0;

for( int dayCount =1; dayCount <=7; ++dayCount)
{
```

$\xrightarrow{9}$

```
    weeklySales += dailySales;
}


std:: cout << weeklySales << std::endl;
```

When to Use Each

Recall: for loops are count-controlled.

→ we know how many times the loop will execute

THAT SHOULD BE THE ONLY REASON YOU USE A FOR LOOP.

while loops are for when you do NOT know how many times the loop should execute → just that it should STOP once the condition is broken.

→ Compound tests where there is no pure count control

eg, upper/lower bound; condition

Query 5. What might we say on a do-while loop?

→ when we want the body to execute at least once.

Nested Loops

Def. Nested Loop is a loop which appears inside another.

↳ good for when one wishes to repeatedly perform a repetitive operation.

St.,

```
for (          ) {
    for (      ) {
    }
}
```

- For each iteration of the outer loop, the inner loop exhausts all of its own iterations

e.g.,

```
for (8 times) {
    for (3 times) {
    }
}
```

+ the iterations of the inner loop "go faster" than that of the outer loop

Breaking & Continuing

Recall: the break statement causes an end to the execution of a control statement.

```
e.g., switch ( const. or
               literal )   {
          case :
                 break;
      }
```

In the context of loops, the break statement causes an end to the execution of the loop.

Query 6. What does the aforementioned say about an inner loop?

⇒ It will break out of the inner loop. Because of scope.

Although familiar, the break statement in loops and switches are a tad different.

eg.,

(Query 7.)

```
loop {
    switch (constant/literals) {
        // ... cases ... //
    }
}
```

breaks out of the switch, **NOT** the loop.

The continue ~~statement~~ causes the current iteration to end and the next to begin.

ie, the continue statement serves as an "interrupt" to the current iteration and forces the next one to begin immediately.

Query 8. What happens, then, in a while or do-while loop?

It will move to the testing condition.

In the for loop, execution will move to the updating step. Afterwards, the testing condition is encountered.

Consider. In China, the 4th floor of a hospital is considered bad luck. Write pseudocode for a loop which prints the no. of floors a cancer pt is on, skipping the fourth.

NOTE

You are NOT to use the continue or the break statement to terminate a loop in this course. It makes programs more difficult to debug.

CS 1436.001:

For loops, When to Use Each Type of loop, Running Totals, Nested Loops, and Breaking/Continuing.

Def. A for loop is a count-controlled loop which iterates a set no. of times.

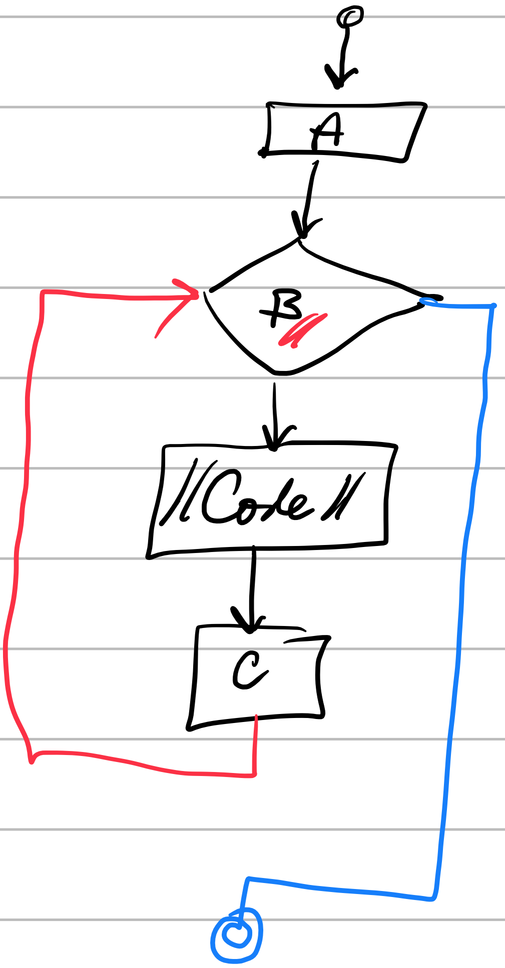
eg.,
```
for (A; B; C) {
    // Code. //
}
```

A := iterator

B := loop condition

C := updating step

Query 1. Is the for loop a pre or post-test loop?



```
if ( Condition ) {                    };
        ⋮
if ( Condition ) {                    };

for (A; B; C) {                       };

for ( A0, A1 ; B ; C ) {
                                  (Query 2)
}
```

```
=> for( int x =1, int y=2; x<=4; x++) {
        std::cout << " sum= " << (x+y);
   }
```

Query 3. Valid? Yes.

```
for( int x=1, int y=2; x<=1000; x++, --y){
      <——————————————> ;
}
```

The loops can be modified further :-

```
→ int num = 1 ;

  for ( ; B ; C ) {

  }
```

B & C MUST relate to num

Query 1.

```
for( ;B; ) {
    C
}
```

header

Running Totals

Def. A running total is that sum of numbers accumulated over the iterations of a loop.

(Cont'd).

Program Example 1.

```
int num, sum = 0;

for (num = 1 ; num ≤ 10 ; num++) {
    sum += num;
}

std::cout << sum << std::endl;
```

Programming Example 2.

```
double dailySales, weeklySales = 0;

for (int dayCount = 1; dayCount ≤ 7 ; ++dayCount) {

    weeklySales += dailySales;
}

std::cout << "Weekly sales: " << weeklySales;
```

Which Loops to Use

for loops are for when we know how many times the loop ought to execute.

While loops are for when no. of executions are not known — just know the loop should STOP upon the loop condition being broken.

→ best for compound tests where pure count control does not hold. (upper/lower bound; another condition).

Query 8. What remarks can be said of the do-while.

We know the do-while will execute at least once.

(contd).

Nested Loops

Def. Nested Loops are loops which appear inside other loops.

↳ good for when one wants to repeatedly perform a repetitive operation,

Ex.,

```
for (          ) {
    <------------->
    for (    ) {
        <----------->
    }
}
```

For each iteration of the outer loop, the inner loop exhausts all of its iterations.

ie,

21 times

```
for ( 3 ) {
    <---------->
    for ( 7 ) {
        <--------->
    }
}
```

(+) The iterations of the inner loop goes faster than the outer loop.

Breaking & Continuing

Recall: the break statement causes an end to the execution of a control structure.

eg,
```
switch( Constant/literals ) {
        case []:
                break;
}
```

In the context of loops, the break statement causes an end to the execution of the loop.

Query 6. What happens in a nested loop?

break will end the inner loop. Because of scope!

Although familiar, the breaks in loops and switches are different.

eg.,
```
loop {
    switch (        ) {
        case []:
            break
    }
}
```

The continue ~~statement~~ causes the current iteration ~~to~~ end and the next to begin.

ie, the continue statement causes an "interupt" to the current iteration and forces a move onto the next one.

In the do-while and while loops, the execution goes straight to the testing condition.

Whereas, in a for loop, execution moves to the updating step. Afterwards, the testing condition is met.

Consider: In China, the 4th floor of a hospital is considered bad luck. Write pseudocode for a loop which prints the current floor a cancer pt is on — skipping the fourth.

You are NOT to use the break OR the continue statement to terminate a loop in this course. It makes programs hard to track and harder to debug #.