Introduction to Programming (CS 101)

Spring 2024

Lecture 5:

for loop, increment/decrement operators, constants

Instructor: Preethi Jyothi

Based on material developed by Prof. Abhiram Ranade

Recap-I (nested `if`): Dangling else problem

What is the output from the following piece of code? The formatting of the statements may not correctly reflect the underlying behaviour.

`Number's more than 10`

No output

`Positive number`

```
#include <simplecpp>

main_program{
  int n = -1;
  if(n < 10)
    if(n > 0)
      cout << "Positive number\n";
  else
    cout << "Number's more than 10\n";

}
```

Recap-II (`while` statement)

What does this program output (in words)?
For `n = 44`, what value of `p` is printed?

44

42

32

64

```
#include <simplecpp>
main_program{

  unsigned int n;
  cin >> n;
  unsigned int p = 1;

  while(p * 2 <= n)
    p *= 2;

  cout << p;
}
```

Recap-III (`while` and `break`)

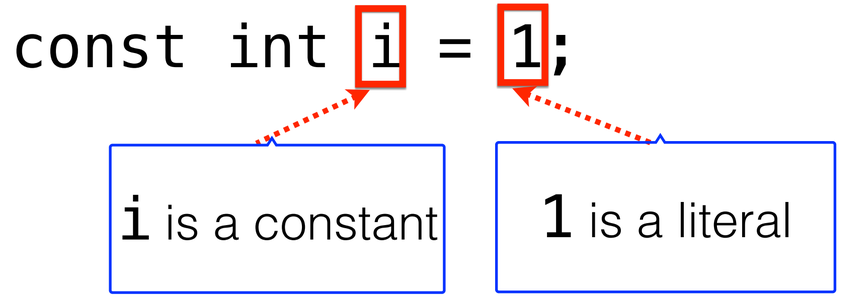
Demo in class and code (guess.cpp) shared on Moodle

Constants

CS 101, 2025

Constants

- A constant is a variable with a fixed value defined before the program runs
- We use the `const` keyword to define constants:



- `const` is an annotation to any type that ensures that it will not be changed
- A statement `i = 5;` after the const definition above will lead to a compiler error
- Why use constant variables instead of just literals themselves?
  - Improved readability
  - Modular if the value needs to be changed; can make the change in just one place

`for` statement

CS 101, 2025

Motivation: `for` statement

- Example: Write a program to print a table of cubes of numbers from 1 to 100

```
int i = 1;

repeat(100) {
  cout << i << "|" << i*i*i << endl;
  i+=1;
}
```

- The idiom above, which is, "do something for every number between x and y" occurs very commonly

- The `for` loop, with its syntax, makes it easy to implement this idiom

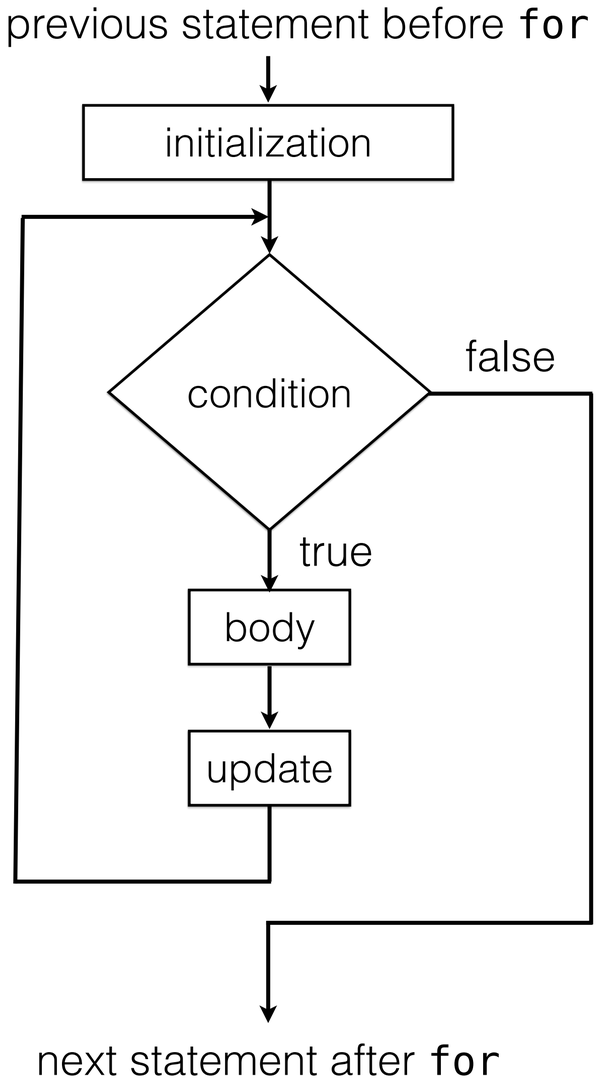
`for` syntax and semantics

- Syntax:

```
for(initialization; condition; update) {
    body
}
```

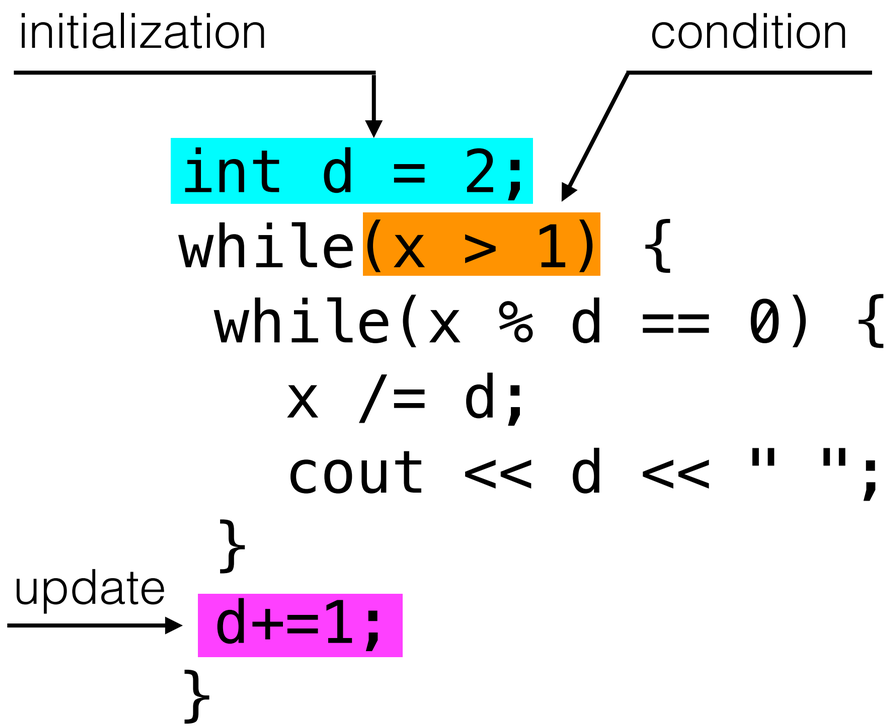
- Example:

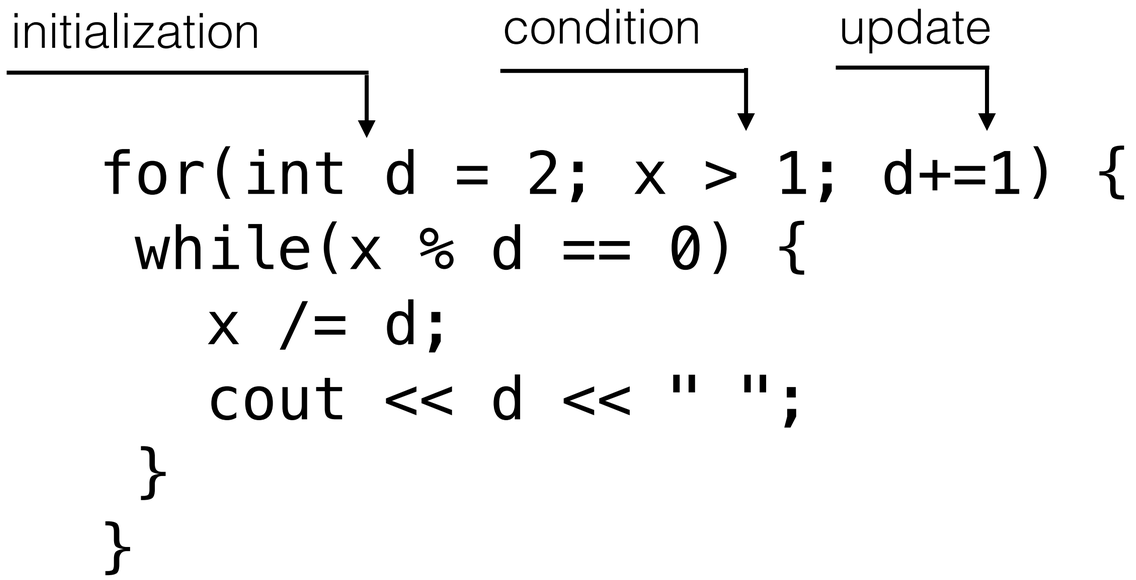
```
for(int i = 1; i <= 100; i++) {
  cout << i << " " << i*i << endl;
}
```



- Semantics:
  - Before the first iteration of the loop, *initialization* is executed
  - Within each iteration:
    - If *condition* evaluates to false, the loop terminates.
    - If *condition* evaluates to true, **body** is executed, followed by *update*. Then, the next iteration begins.

`while` and `for`





- Note that a new variable `d` (`int d`) is defined in the initialization. `d` is accessible within the loop body, i.e. the scope of `d` is the `for` loop's body. `d` is accessible within "condition" and "update" as well.
- Note that the scope of `d` in the code on the left extends beyond the outer while loop. However, the scope of `d` in the code on the right is only within the for loop; `d` is not accessible outside the `for` loop.

`for` examples

- `repeat(n) { ... }` can be replaced with:

```
for(int i = 0; i < n; i+=1){ ... }
```

OR

```
for(int i = n; i > 0; i-=1){ ... }
```

- `for( ; ;)` is allowed. That is, empty condition, empty initialization and empty update is allowed.
  - This would be an infinite loop, like `while(true)`
  - Would need a `break` statement to get out of the loop

Increment, decrement operators

- `for(int i = 0; i < n; i+=1) { ... }` is more commonly written as

```
for(int i = 0; i < n; i++) { ... }
```

OR

```
for(int i = 0; i < n; ++i) { ... }
```

  using the increment (`++`) operator.

- There is similarly a `--` decrement operator (`i--`, `--i`)

- Both `++i` and `i++` in the `for` loop above work as shorthand for `i = i + 1`

... with an important caveat

Increment, decrement operators

- An important difference between the pre-increment (`++i`) and post-increment (`i++`) operators:
  - `++i` will first increment `i` and evaluate to the incremented value
  - `i++` will evaluate to the original value of i prior to incrementing
  - Similarly, the pre-decrement (`--i`) and post-decrement (`i--`) operators

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

output

This would output 1 to 10

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

output

This would output 0 to 9

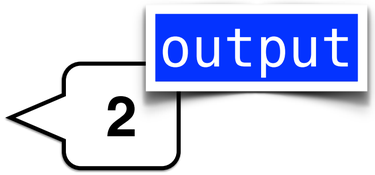
Increment, decrement operators

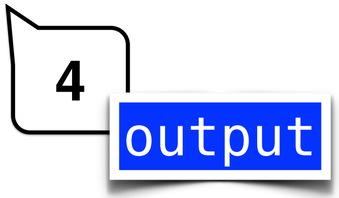
- What is the output of this program?

```
int x = 0, y = 0;

for(int i = 0; i < 3; i++) {
    x += i++;
}

for(int i = 0; i < 3; ++i) {
    y += ++i;
}
cout << x << "\n";
cout << y << "\n";
```



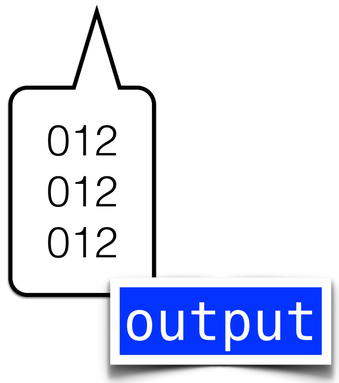


nested for example

- What does this program do?

```
#include <simplecpp>

main_program {
    for(int i = 0; i < 3; ++i) {
       for(int j = 0; j < 3; j++) cout << j;
       cout << "\n";
    }
}
```



Another `for` example

- What does this program do?

```
char c;
unsigned int n;

for(cin >> c; c<'0' || c>'9'; cin >> c);
for(n=0; c>='0' && c<='9'; n=n*10+(c-'0'), cin>>c);
```

Note the empty body of this `for` loop.

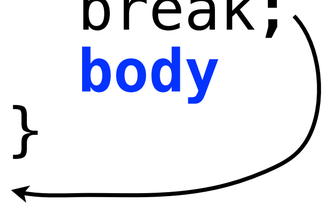
Note the `,` (comma) operator. Different from `,` used as a separator (E.g., `int i, j;`). Expressions separated by a comma are evaluated left-to-right.

`output`

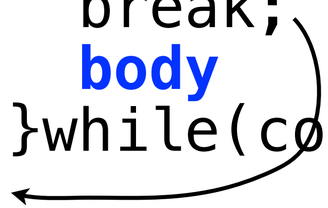
The first for loop will ignore non-digit letters till the first digit is encountered. Then, a number is accumulated from the digits.

`break` across different loop structures

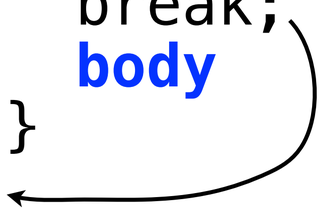
```
while(condition) {
   // ...
   break;
   body
}
```



```
do {
   // ...
   break;
   body
}while(condition)
```



```
for(initialization;condition;update) {
   // ...
   break;
   body
}
```



- Note consistent behaviour of `break` across while, do-while and for loop structures:
  - Causes a jump to the statement immediately following the enclosing loop

- `break` can appear only within a loop body or within `switch` (case) blocks
- break within nested loops will terminate only the enclosing loop (and not break out of all nested loops)

`continue` across different loop structures

```
while(condition) {
   // ...
   continue;
   body
}
```

```
do {
   // ...
   continue;
   body
}while(condition)
```

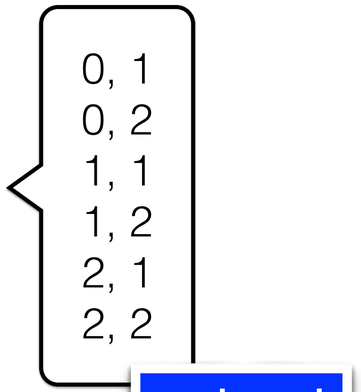
```
for(initialization;condition;update) {
   // ...
   continue;
   body
}
```

- Note consistent behaviour of continue across while, do-while and for loop structures:
  - Causes a jump to the end of the loop **body**

`continue` in a nested loop

- What is the output of this program?

```
main_program {
 for(int i = 0; i < 3; i++) {
   for(int j = 0; j < 3; j++) {
     if(j == 0)
       continue;
     cout << i << "," << j << endl;
   }
 }
}
```



Next class: Internal Representations of data types

CS 101, 2025