Introduction to Computer Science
Lecture 5

Nazareno Aguirre

(based on material by Guillaume Hoffmann)

Review

Increment and decrement operators

```
int a, b = 0, c = 0;
a = ++b + ++c;
printf("%d %d %d\n", a, b, c);
a = b++ + c++;
printf("%d %d %d\n", a, b, c);
a = ++b + c++;
printf("%d %d %d\n", a, b, c);
a = b-- + --c;
printf("%d %d %d\n", a, b, c);
```

What will this program print out?

Review

Iteration

```
#include <stdio.h>
int main() {
    int h; scanf("%d", &h); int b; int z;
        z=1;
    b=z;
    while (!(z>=h+1)) { z=z+1;b=(z-1)*b; }
    printf("%d\n", b);
}
```

What does this program do?

Review

Iteration

```c
#include <stdio.h>

int main() {

    int n;
    scanf("%d", &n);

    int factorial = 1;
    int i = 1;
    while (i <= n) {
        factorial = factorial * i;
        i++;
    }
    printf("%d\n", factorial);

    return 0;
}
```

Indent correctly.
Choose meaningful variable names.
Favor good forms of iteration.
Use comments!

```c
/* Computes the factorial of a natural number n.
 * The factorial of n is the result of the product (1 * 2 * ... * n).
 * The argument is received from standard input.
 * The result is printed out on standard output.
 *
 * Author: John Doe
 * Date: April 18th 2023
 * Version: 0.1
 */
```

Today's topics

- Assignment operators
- **Definite iteration (“for” loops)**
- **Computing sums and products of sequences**
- The return statement in main()
- The comma operator

Assignment operators

- In addition to `=`, there are other assignment operators, such as `+=` and `-=`.
- An expression such as:

```
k = k + 2;
```

  adds `2` to the (old) value of `k`, and assigns the result to `k`. Also, the expression as a whole will have that value (the value of the right hand side expression).

- The expression:

```
k += 2;
```

accomplishes the same task!

Assignment operators

The different assignment operators have the following form:

```
variable op= expression
```

where `op` is an operator. This is equivalent to

```
variable = variable op (expression)
```

Assignment operators										
=	+=	-=	*=	/=	%=	>>=	<<=	&=	^=	|=

Assignment operators - An Example

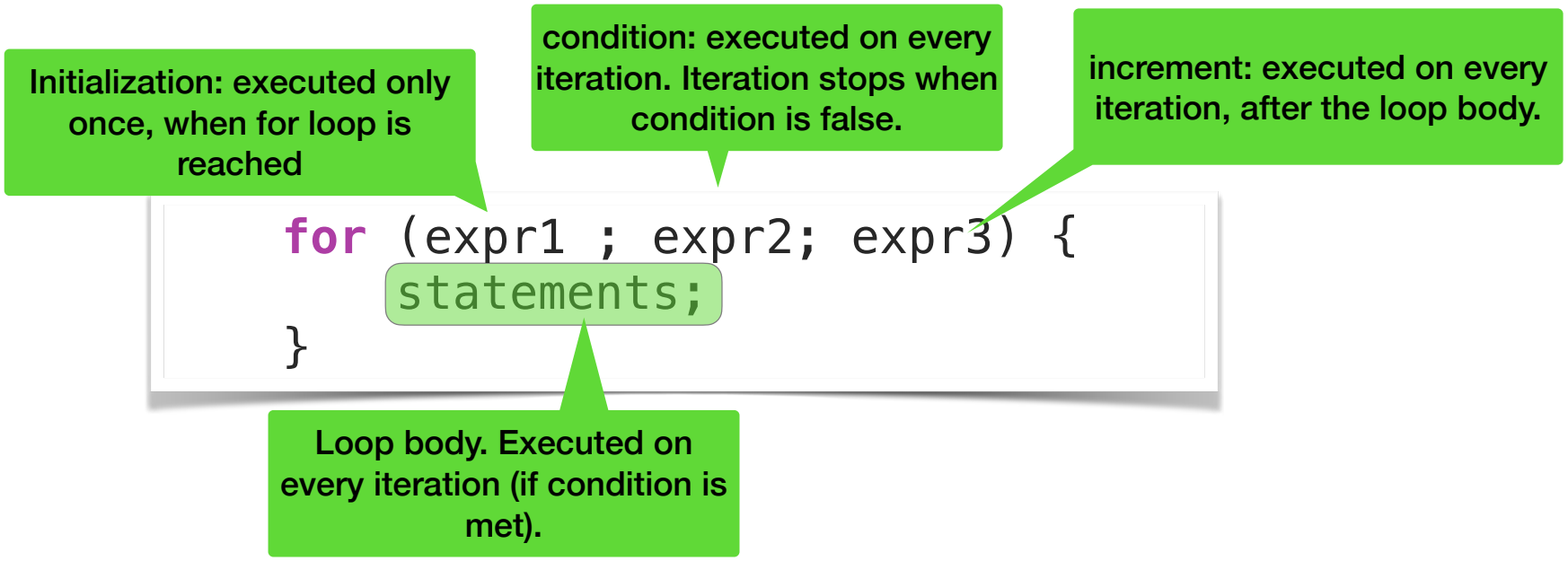
```c
#include <stdio.h>

/* Computes the factorial of a natural number n.
 * The factorial of n is the result of the product (1 * 2 * ... * n).
 * The argument is received from standard input.
 * The result is printed out on standard output.
 *
 * Author: John Doe
 * Date: April 18th 2023
 * Version: 0.1
 */
int main() {

    int n;
    scanf("%d", &n);

    int factorial = 1;
    int i = 1;
    while (i <= n) {
        factorial *= i;
        i++;
    }
    printf("%d\n", factorial);

    return 0;
}
```

The For Statement

- It provides an additional mechanism to define iteration
  - It can be reduced to “while” loops (while loops are more expressive)
- For loops correspond to a form of iteration known as **“definite iteration”**
  - In definite iteration, the number of times that the loop body is executed is determined
- Very useful for certain iteration patterns, e.g., sequence summations and products.
- In C, it has the following syntax:

```
for (expr1 ; expr2; expr3) {
    statements;
}
```

The For Statement



For Loop Example

```c
#include <stdio.h>

/* Computes the factorial of a natural number n.
 * The factorial of n is the result of the product (1 * 2 * ... * n).
 * The argument is received from standard input.
 * The result is printed out on standard output.
 *
 * Author: John Doe
 * Date: April 18th 2023
 * Version: 0.1
 */
int main() {

    int n;
    scanf("%d", &n);

    int factorial = 1;
    for (int i = 1; i <= n; i++) {
        factorial = factorial * i;
    }
    printf("%d\n", factorial);

    return 0;
}
```

For Loops vs While Loops

- For loops can also be written as while loops (while loops are more expressive):

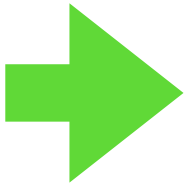
```
for (expr1 ; expr2; expr3) {
    statements;
}
```

is equivalent to

```
expr1;
while (expr2) {
    statements;
    expr3;
}
```

For Loop to While Loop Reduction

```
int factorial = 1;
for (int i = 1; i <= n; i++) {
    factorial = factorial * i;
}
```



```
int factorial = 1;
int i = 1;
while (i <= n) {
    factorial = factorial * i;
    i++;
}
```

When and How to use a For statement

- The **for loop** is best used when the number of times the loop will iterate is determined, and can be expressed in relation to iteration variables.
    - It can also favor readability, with initialization/condition/increment all defined at the top of the loop.
- Iteration variable should not be modified in the loop body
    - In that way, we maintain the good practice of “advancing” iteration at the end of the loop body
- Some really **bad practices**:
    - Modify the iteration variables in the loop body
    - Write conditions with side effects
    - Put statements not related to the iteration in the initialization

Manual Execution Example

```
int a = 0;
for (int i = 0; i < 2; i = i+1) {
    a = a + 2;
}
```

	a	i	condition
int a = 0	0		
Int i = 0		0	
i < 2			true
a = a + 2	2		
i = i + 1		1	
i < 2			true
a = a + 2	4		
i = i + 1		2	
i < 2			false

The ‘Continue’ Statement in a For Loop

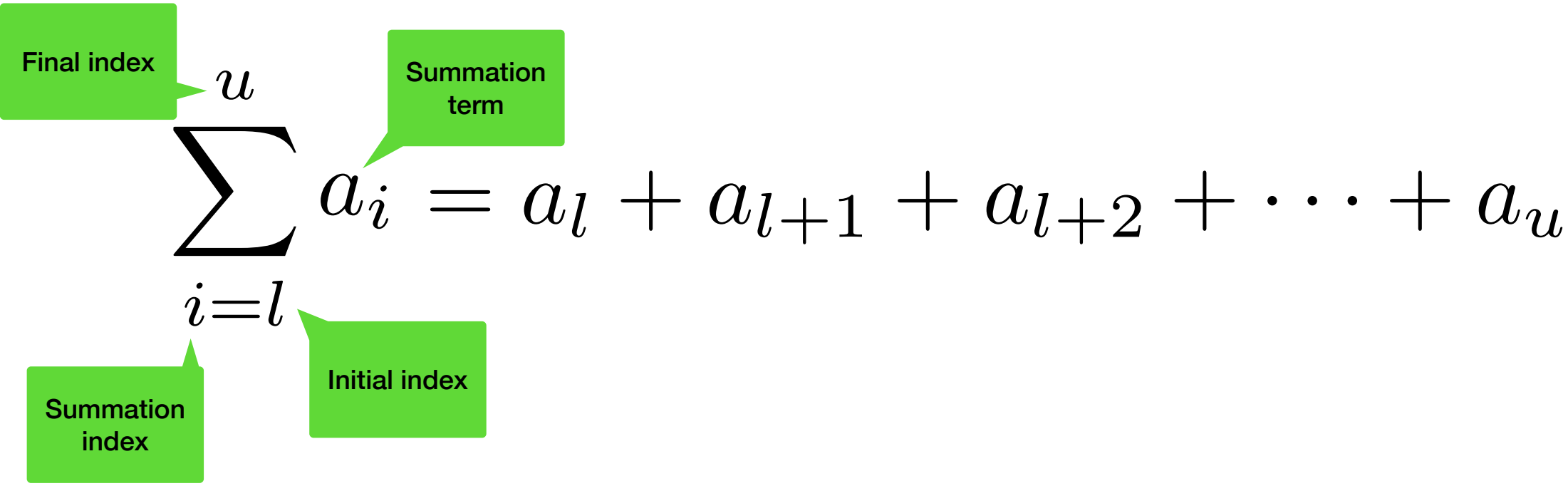
- We have already discussed that the use of “loop breaking” statements such as `continue` and `break` should be avoided
  - It is nevertheless useful to know how these work
- The `continue` statement in a **while** loop makes the execution “skip” the remainder of the loop body, i.e., to jump directly to loop condition.
  - In a **for** loop, it makes the execution jump to the **increment** of the loop. Then, the condition is evaluated.

```
int factorial = 1;
for (int i = 0; i <= n; i++) {
    if (i == 0) continue;
    factorial = factorial * i;
}
printf("%d\n", factorial);
```

It will skip the rest of the loop body, but execute “i++” before checking the loop condition (i <= n)

Sigma Notation (sequence summation)

A mathematical expression that represents the **sum of terms of a sequence** within a certain range.



Summation Examples

$$\sum_{i=1}^{5} i = 1 + 2 + 3 + 4 + 5 = 15$$

$$\sum_{j=2}^{1} j^3 = 0$$

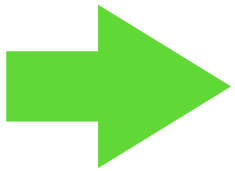
$$\sum_{j=0}^{4} j^2 = 0^2 + 1^2 + 2^2 + 3^2 + 4^2 = 30$$

$$\sum_{j=1}^{100} 1 = 100$$

Summations and their implementation

Summations can be straightforwardly mapped into programs, using for loops

$$\sum_{i=l}^{u} a_i$$

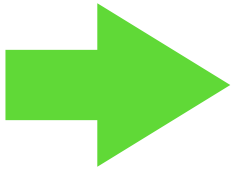


```
int sum = 0;
for (int i = l; i <= u; i++) {
    sum = sum + a(i);
}
```

expression that represents the term

Summations and their implementations - Example

$$\sum_{j=0}^{4} j^2$$



```
int sum = 0;
for (int j = 0; j <= 4; j++) {
    sum = sum + (j * j);
}
```

The `return` statement

- The return statement is used to communicate return values between programs
  - When modularizing programs in functions (we'll see that later), return statements are used to communicate back the results of function computations
- In our simple programs, with just a main() routine, we often see a **`return 0`** statement
  - What happens if we put the `return 0;` statement elsewhere?
    - Program exits as soon as the return statement is reached.
- What happens if we change the `0` integer constant by another one, like `1`?
  - As a convention, return 0 is used to indicate that the program has successfully achieved its goal (returning a non-zero value will indicate that there has been a problem in the computation)

The Comma Operator

- The comma operator is a binary operator with expressions as operands.
- In a comma expression of the form `expr1, expr2`

  `expr1` is evaluated first, and then `expr2`.
- The comma expression as a whole has the value and type of its right operand.

  Example:

  ```
  a = 0, b = 1
  ```

  If `b` is an `int`, then this comma expression has value `1` and type `int`.

The Comma Operator

- The comma operator sometimes gets used in `for` statements.
- It allows multiple initializations and multiple processing of indices.
- For example, the code:

```
for(sum=0, i=1; i <= n; ++i)
    sum +=i;
```

can be used to compute the sum of integers from `1` to `n`.

The Comma Operator, a not-so-recommendable example

- Carrying the idea further, we can stuff the entire loop body into the for parentheses.

```
for(sum=0, i=1; i <= n; sum +=i, ++i);
```

- But not as:

```
for(sum=0, i=1; i <= n; ++i, sum +=i);
```

Because it would make `++i` be evaluated first and `sum` would have a different value.

These uses of comma should be avoided.

Readability and clarity are seriously affected when abusing comma statements