Introduction to Computer Science
Lecture 4

Nazareno Aguirre

(based on material by Guillaume Hoffmann)

Warmup

What is the effect of the following statement?

```
x = (y = 2) + (z = 3);
```

Today's topics

- Increment and decrement operators
- The || and && operators: short circuit evaluation
- The conditional operator ? :
- Break and continue statements

Increment and decrement operators

- The increment operator `++` and the decrement operator `--` are *unary* operators.
  - Unlike + and – which are *binary* operators.
- They can be used as prefix and postfix.
- Suppose `var` is a variable of type int:
  - Both `++var` and `var++` are valid expressions
    - `++var` uses `++` as a prefix operator
    - `var++` uses `++` as a postfix operator

Increment and decrement operators

- Increment and decrement operators can only be applied to variables, not to constants or ordinary expressions:

```
777++                   /* constants cannot be incremented */
++(a * b - 1)           /* expressions cannot be incremented */
```

Increment and decrement operators

- Expressions `++i` and `i++` have an effect **and** a value
- Effect: they cause the value stored in `i` to be incremented by 1
- Value: the result of the evaluation of the expression
  - ++i causes the stored value of i to be incremented, with the expression having as associated value its new (incremented) value, stored in i.
  - i++ causes the stored value of i to be incremented too, but the expression has as associated value its original value of i (prior to the increment).

```c
int a, b;
int c = 0;
a = ++c;
b = c++;
printf("%d %d %d", a, b, ++c)        /* 1 1 3 is printed */
```

Increment and decrement operators

- `--i` and `i--` work in a similar way
- The operators `++` and `--` cause the value of a variable in memory to be changed.
    - We say that operators `++` and `--` have *side effects*: not only do these operators yield a value, they also *change* the stored value of a variable in memory.
    - Not all operators do this. For example, the expression `(a + b)` leaves the values of variables `a` and `b` unchanged.

Increment and decrement operators

- Often, using `++`/`--` in either prefix or postfix position will produce the same result.
  - I.e., statements: `++i;` and `i++;` are often equivalent to `i = i + 1;`
  - ...unless they are used inside a condition or a greater expression!
- In simple situations, `++` and `--` provide a short notation for the incrementing and decrementing a variable.
- In other situations, careful attention must be paid as to whether prefix or postfix position is desired.

Increment and decrement operators

Notice the difference between these two loops (assume `i` contains some positive value):

```
while (i--)
    printf("%d ",i);
```

```
while (--i)
    printf("%d ",i);
```

Short-Circuit Evaluation

- In the evaluation of operands of `&&` and `||`, the evaluation process starts from the left and **stops** as soon as the outcome TRUE or FALSE is known.
- Consider the evaluation of expression: `expr1 && expr2`
  - If `expr1` has value zero/FALSE, then evaluation of `expr2` does not occur.
- Consider the evaluation of expression: `expr1 || expr2`
  - If `expr1` has value non-zero/TRUE, then evaluation of `expr2` does not occur.

Short-Circuit Riddle

```c
int a = 0, b = 0, x;

x = 0 && (a = b = 777);
printf("%d %d %d\n", a, b, x);
x = 777 || (a = ++b);
printf("%d %d %d\n", a, b, x);
```

```c
int a = 0, b = 0, x;

(x = 0) && (a = b = 777);
printf("%d %d %d\n", a, b, x);
(x = 777) || (a = ++b);
printf("%d %d %d\n", a, b, x);
```

What do these programs print on the screen?

Short-Circuit Exercise (harder)

In the following code, assume that the values of `i` and `j` are not changed in the body of the loop.

```c
printf("Input two integers: ");
scanf("%d", &i);
scanf("%d", &j);
while (i*j<0 && ++i != 7 && j++ != 9) {
    // do something...
}
```

Can this ever lead to an infinite loop?

The Conditional Operator

The conditional operator `?:` is a ternary operator. It takes as operands three expressions:

```
expr1 ? expr2 : expr3
```

where `expr1`, `expr2` and `expr3` are expressions. In the above construction,

- `expr1` is evaluated first.
  - If `expr1` is non-zero (true), then `expr2` is evaluated, and the result of the evaluation is the value of the conditional expression.
  - If `expr1` is zero (false), then `expr3` is evaluated, and the result of this evaluation is the value of the conditional expression.

The Conditional Operator

In many situation, the conditional operator enables us to write more concise and readable programs:

```
if (y < z)
    x = y;
else
    x = z;
```

is equivalent to

```
x = (y < z)? y : z;
```

The Conditional Operator

```c
int main() {
    int input;
    printf("How many cherries do you want?\n");
    scanf("%d", &input);
    printf("So you want %d cherr%s.\n", input, (input == 1) ? "y" : "ies");
    return 0;
}
```

The break and continue statements

- Normally, loops can only exit when the loop condition is false.
- Also, the whole body of the loop is executed at each iteration.
  - `break` and `continue` provide ways of exiting a loop or jumping to the condition of the loop directly from any place within the loop body.
  - **`break`**:
    - A break causes the innermost enclosing loop to be exited immediately.
  - **`continue`**:
    - A continue makes execution jump to the loop condition.

Break Statement. An Example

A typical use of break. What would otherwise be an infinite loop is made to terminate upon a given condition tested by the if expression:

```
while (1) {
    scanf("&d", &x);
    if (x < 0)
        break;    /* exit loop if x is negative */
    printf("%d", x*x);
}
/* break jumps here */
```

Break Statement. Another Example

```
printf("Please enter a number between 0 and 100\n");
while (1) {
    scanf("%d", &x);
    if (x >= 0 && x <= 100)
        break;
    printf("Sorry. Please enter a number between 0 and 100\n");
}
```

The continue statement

The `continue` statement stops the current iteration, and causes the next iteration of the loop to begin immediately.

- It can only be used inside iteration statements.

```c
int i = 0;
while (i < TOTAL) {
    c = getchar();
    if (c >= '0' && c <= '9')
        continue;
    /* process other characters */
    ++i;
}
```

Ending Remarks

- We covered a number of important topics today:
  - C expressions have a value and may have side effects
  - Some widely used operators (e.g., increment/decrement) have side effects, so we must be careful when using them in expressions
    - (in particular, when used as part of expressions with short-circuit operators)
  - Some control statements allow us to alter the normal control flow in loops (break, continue)
    - They should usually be avoided, as they lead to programs with poor structure and difficult to reason about
    - In some situations, they can help us write cleaner and more readable code
      - But this is rare!