CS100 Recitation 2

GKxx

Februrary 28, 2022

Contents

++ and --

Increment and decrement operators

- Both `i++` and `++i` increases the value of `i` by `1`.
- What are the values of `i`, `j` and `k` after the following code is executed?

```
int i = 42;
int j = ++i;
int k = i++;
```

`i == 44, j == 43, k == 43`.

++ and --

- What are the values of `j` and `k`?

```
int i = 42;
int j = ++i, k = i++;
```

`j == 43, k == 43.`

- What's the output of the following code?

```
int i = 42;
printf("%d, %d", ++i, i++);
```

Undefined behavior!

++ and --

The prefix increment operator:

1. Increases the value of the variable.
2. Returns the variable.

The postfix increment operator:

1. Saves the original value of the variable.
2. Increases the value of the variable.
3. Returns the original value that has been saved.

- More about the difference between them will be discussed in the C++ part.

Relational Operators

Relational Operators: `<`, `<=`, `>`, `>=`, `==`, `!=`.

- What is the return-type of these operators?
  Unfortunately it is `int` instead of `bool`, due to the problematic definitions of `true` and `false` before C23.
  In C++, it is undoubtedly `bool`.
- What's the result of the expression `a < b < c`?
  It behaves as expected in Python, but not in C.

Arithmetic Operators

Arithmetic Operators: `+` (unary/binary), `-` (unary/binary), `*`, `/`, `%`, as well as bitwise operators `&`, `^`, `|`, `~`, `<<`, `>>`.

- If at least one of the operands is floating-point, the other integer operand, if any, will be converted to the same floating-point type. (More about type conversion will be discussed later.)
- Division for integers:
  - Rounded in implementation-defined direction. (Until C99)
  - Truncated towards zero. (Since C99)
  - e.g. `3 / -2 == -1` (since C99)
- Remainder: `(a / b) * b + a % b == a` is always true.
- Bitwise operators will be discussed in the next recitation.

Compound Assignment Operators

Compound assignment operators: `+=`, `-=`, `*=`, `/=`, `%=`, `<<=`, `>>=`, `&=`, `|=`, `^=`.

- '`a = a op b`' is the same as '`a op= b`'.
- Practice to use them more, as they are clear and increases readability.

Conditional Operator

Conditional operator: `cond ? exprT : exprF`.

- The evaluation order is determined!
  - `cond` will be evaluated first. If `cond` evaluates `true`, `exprT` will be evaluated, otherwise `exprF` will be evaluated.
  - Only one of `exprT` and `exprF` will be evaluated.
- It is suggested to use it for simple occasions like '`a < b ? a : b`'.
- Nested conditional expressions reduces the readability sharply!
  - `a < b ? (a < c ? a : c) : (b < c ? b : c)`

Contents

scanf and printf

For the authoritative reference, view it on `https://en.cppreference.com/w/c/io/fscanf` and `https://en.cppreference.com/w/c/io/fprintf`.
Some common issues:

- You should always make sure that the format string and the variables **match** each other, especially in `scanf`!
- More details about mismatch will be discussed after you have learned pointers and conversions.
- Any whitespace character consumes all available consecutive whitespace characters. So the statement `scanf("%d\n", &a);` keeps waiting for the next non-whitespace character.

scanf and printf

- Preceeding whitespaces will be ignored when matching conversion specifiers (beginning with '%'), but **will not be ignored** when matching other characters!
  - `scanf("(%d,%d)", &a, &b);`
  - If '`(3, 2)`' is inputted, the space before '2' will be ignored.
  - What about this?

```
for (int i = 0; i < n; ++i)
    scanf("(%d,%d)", &a, &b);
```

If the data is inputted line-by-line, the first input stops at the first newline character.
Then that newline character is read in the second iteration, which does not match '('. Thus a failure occurs.
To solve this, write `scanf(" (%d,%d)", &a, &b);`.

scanf and printf

- `scanf("%c", &a);` does not ignore preceeding whitespaces! (You may have a try on your own.)
- `scanf` returns an `int` value, denoting the number of receiving arguments successfully assigned, or `EOF` (-1) if input failure occurs before the first receiving argument was assigned.
- You can use the return-value to detect failure of `scanf`.
  - `int r = scanf("(%d,%d)", &a, &b);`
  - If two integers are read and assigned successfully, `r` will be assigned 2.

Contents

if-else

- Which `if` does the `else` match?

```
if (a == b)
    if (c < d)
        do_something();
else
    do_another_thing();
```

- Dangling `else`!

if-else

- Use `else` properly to avoid repeated calculation, and also make your code more robust.

```
if (b == 0)
    printf("Error: Division by zero!\n");
if (b != 0) // Better to use ‘else’ here!
    printf("The answer is %lf\n", a / b);
```

if-else

Interesting fact

The ‘)’ is used to separete the condition and the statement. The only reason for writing ‘(’ is to match the ‘)’.

`if-else` in Python:

```
if condition:
    statement
```

In Pascal:

```
if condition then
    statement
```

for loop


```
for (init-expression; condition; expression)
    statement
```

- Since C99, variable declaration is allowed in the 'init-expression' part.

```
for (int i = 0; i < n; ++i)
    do_something();
```

- The variable `i` is declared in the `for` loop, and destroyed immediately the loop ends.

```
for (int i = 0; i < n; ++i)
    do_something();
printf("%d\n", i); // Error: i was not declared in
    this scope.
```

The self-teach algorithm

Algorithm
self-teach:
1 read problem
2 attempt solution
3 skim book solution
4 if attempt failed
goto 1
else goto next problem

Unfortunately, that algorithm can put you in an infinite loop.

Suggested patches:

0 set $c \leftarrow 0$
3a set $c \leftarrow c + 1$
3b if $c = N$
goto your TA

—*E. W. Dijkstra*

(In *Concrete Mathematics*, Chapter 5 Section 2.)

Contents

Variable naming

- `int num_of_student;`
- `int numOfStudent;`
- `#define SIZE 128`
- The names of normal variables or functions are recommended to begin in lower case.
- Macros are often named in upper case.
- Similar rules apply to the names of folders and files!

Variable declaration

- In C89, variables declarations are only allowed at the beginning of blocks.
- This requirement is removed since C99.

> **Advice: Define Variables Where You First Use Them**
>
> It is usually a good idea to define an object near the point at which the object is first used. Doing so improves readability by making it easy to find the definition of the variable. More importantly, it is often easier to give the variable a useful initial value when the variable is defined close to where it is first used.

Variable initialization

You must remember the following firmly!

- Local non-static variables, if not explicitly initialized, will be default initialized, that is, initialized with an underined value.
- Global variables or local static variables, if not explicitly initialized, will be value initialized, that is, initialized with zero. (character types and integer types: `0`; floating-point types: `0.0`; pointers: `NULL`).
- Arrays are initialized element-by-element according to the same rule above.
- Additional rule for arrays: For local non-static arrays, if an explicit initializer is provided but it does not cover all the elements of the array, those elements not covered will be value initialized!