2. Expressions and flow control

- Expressions
- Selection
  - *if* statements
  - *switch* statements
- Iteration
  - *while* loops
  - *do* loops
  - *for* loops

Expressions

- Mostly like Python, except:
  - No exponentiation operator (`**`)
  - '/' on ints behaves like Python's '//'
  - Logical operators different:

    and ⟶ `&&` *or* ⟶ `||` *not* ⟶ `!`
  - `++` and `--` operators for increment/decrement
    - Use before the variable for pre-increment/decrement, e.g. `i = ++j;`
    - Use after the variable for post-increment/decrement, e.g. `i = j--;`
  - Assignment operator '=' can be used anywhere in expressions
    - e.g. `i = (j = 2) + (k = 10);`
- But ENCE260 style rules (with some exceptions: see labs):
  - Use `++`/`--` only in isolation, i.e. as shorthand for `i += 1` etc
  - Use the assignment operator only in "simple" assignment statements
    - `i = j = k = 0;  // OK`

Statements

- The following are legitimate statements in C:
  - *Expression*
  - *if_statement*
  - *switch_statement*
  - *while_statement*
  - *do_statement*
  - *for_statement*
  - *return_statement*
  - *break and continue statements (and one other unmentionable one)*
  - { declaration | statement … } // A compound statement or "block"

Selection statements

Iteration statements

'…' denotes "zero or more"

- Statements must be terminated with a semicolon

if *statement syntax*

- Syntax in BNF notation ("Backus-Naur Form"):

  if_statement ::= "if" "(" expression ")" statement | "if" "(" expression ")" statement "else" statement

  - ::= means "is defined as"
  - | denotes 'or'
  - Tokens in double quotes are "terminals"
    - i.e. they must appear as is
  - Other tokens are expanded by their own syntax definitions
- Note the parentheses around the condition expression
- Syntax reference:
  - http://cse.csusb.edu/dick/samples/c.syntax.html#Statements

if *statement semantics*

- The expression is evaluated. If it is non-zero, the body of the *if* is executed. If not and there is an *else* clause, that is executed.
- Like Python, C has a weak idea of booleans.
  - 0 is *false,* anything else is *true*
- Type *_Bool* is an *unsigned int* restricted to {0, 1}.
  - `#include <stdbool.h>` defines *bool* as equivalent to *_Bool,* and the literals *true* and *false* (1 and 0 resp.)

```
bool isBig = 300; printf("%d\n", isBig); // Prints 1!
```

  - In C99 but not C89

if statement style rules

- ALWAYS lay out on multiple lines, using 4-space indentation and braces around the conditional statement(s)
  - Think Python!
  - CodeRunner style checker now **requires** this
    - Uses *astyle* program
- For example:

```
if (c != EOF) {
    puts("Still going");
} else {
    puts("End of file reached");
}
```

Using the astyle *program*

- Many different C layout standards
  - Use whatever your company says
  - In ENCE260 the company says *1tbs*
    - The "one true brace style"
    - Variant of K&R
- Linux program *astyle* fixes broken layout (mostly)
  - Command *astyle --style=1tbs --indent-labels filename*
  - Done with 2 keystrokes in *geany* (once you've set it up right)
  - See lab 2 "Laying out your code"
- All CodeRunner submissions in ENCE260 must be formatted by *astyle*
  - There is no excuse for badly indented code

nested ifs

- Lay out as …

```
if (a == 0) {
    puts("Not a quadratic");
} else {
    discrim = b * b - 4.0 * a * c;
    if (discrim < 0) {
        puts("Roots are imaginary");
    } else {
        root1 = …  // etc
    }
}
```

An exception (pseudo `elseif`*)*

- Layout multiway *if* statement as:

```
if (c >= 'a' && c <= 'z') {
    puts("Lower case alphabetic");
} else if (c >= 'A' && c <= 'Z') {
    puts("Upper case alphabetic");
} else if (c >= '0' && c <= '9') {
    puts("Numeric");
} else {
    puts("Not alphanumeric");
}
```

NB: no explicit *elif* or *elseif* in C

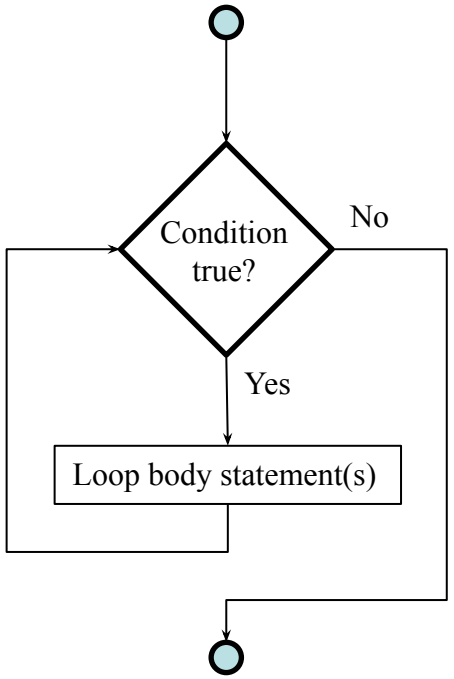
Switch statement

```
switch ( expression ) {
    case constant-expression1 :
        // statements executed if the expression equals the
        // … value of this constant-expression
        break; // Omit this and it "falls through" to the next!

    case constant-expression2 :
        // statements executed if the expression equals the
        // … value of this constant-expression
        break;

    default :
        // statements executed if expression does not equal
        // … any case constant-expression
}
```

Rarely useful. Not usually recommended.

while loops

- Syntax:
  - while_statement ::= "while" "(" expression ")" statement
- As for *if* statement
  - condition expression must in parentheses.
  - Style rules:
    - lay out in multiple lines
    - use braces around loop body statement(s)

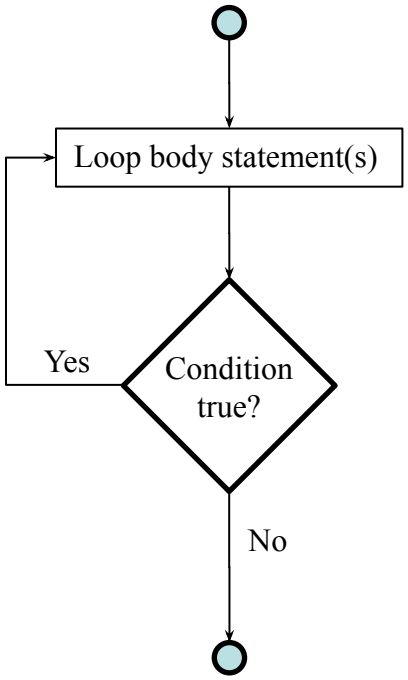



```
int i = 10;
while (i > 0) {
    printf("%d\n", i);
    i--;
}
```

do ... while loop

- A mostly-useless variant of a normal *while* loop

```
int i = 10;
do {
    printf("%d\n", i);
    i--;
} while (i > 0);
```



for loop

- A generalisation of *while*
  - NB: **not** like Python's *for*
- Syntax:
  - for_statement::=

"for" "("expression ";" expression ";" expression")" statement

initialisation *while* condition update loop body

BUT *STYLE RULE*: use only for simple "counted" loops

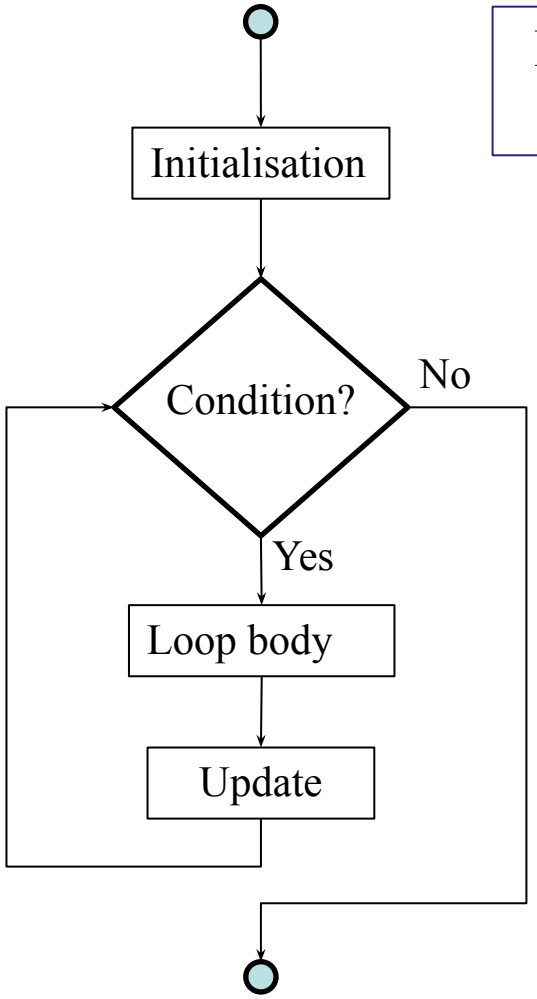
```
i = 10;
while (i > 0) {
    printf("%d\n", i);
    i--;
}
```

=

```
for (i = 10; i > 0; i--) {
    printf("%d\n", i);
}
```

NB: update expression executed *after* loop body statement

for *loop: flowchart*



REMEMBER: use only for simple "counted" loops

Note the update happens on the last iteration, too.