The C Programming Language
Chapter 3

(material from Dr. Michael Lewis, William & Mary Computer Science)

Overview

- Booleans
- Comparison Operators
- Conditional Expressions
- if Statement
- while/for/do Statements
- switch Statement

Python vs. C vs. C++ vs. Java

For the most part the flow control in C is similar to that in Python, though C has a few constructs that Python lacks.

	Python	C	C++	Java
booleans:	`True, False`	`true, false` or non-zero, zero	same as C	`true, false`
logical AND:	`and`	`&&`	same as C	same as C
logical OR:	`or`	`||`	same as C	same as C
logical NOT:	`not`	`!`	same as C	same as C
comparison operators:	`<, <=, ==, !=, >=, >`	same	same[1]	same
identity	`is`			
membership	`in`			
conditional expression	`expr2 if expr1 else expr3`	`expr1 ? expr2 : expr3`	same as C	same as C
conditionals	`if expr:`	`if (expr) {}`	same as C	same as C
	`elif expr:`	`else if (expr) {}`	same as C	same as C
	`else:`	`else {}`	same as C	same as C
iteration	`while expr:`	`while (expr) {}`	same as C	same as C
	`for i in iterable:`		range for: `for (i : object)`	foreach: `for (i : object)`
		`for (expr1; expr2; expr3) {}`	same as C	same as C
		`do {} while (expr);`	same as C	same as C
	`break`	`break;`	same as C	same as C + break and branch
	`continue`	`continue;`	same as C	same as C + continue and branch
choice		`switch`	same as C	same as C
branch		`goto label;`	same as C	
increment/decrement by 1		`++`, `--`	same as C	same as C

[1] The 2020 revision of C++ added a three-way comparison operator `<=>`, also known as "the spaceship".

Booleans

- in C, true represented by 1 and false by 0

```
cat -n ch03/booleans.c
```

```
 1  #include <stdio.h>
 2
 3  int main(int argc, char **argv)
 4  {
 5    if (1) {
 6      printf("1 is true!\n");
 7    }
 8
 9    if (0) {
10      printf("The int 0 is false!\n");
11    }
12
13    if (3.14) {
14      printf("The double 3.14 is true!\n");
15    }
16
17    if ('a') {
18      printf("The char 'a' is true!\n");
19    }
20
21    if (-1) {
22      printf("The int -1 is true!\n");
23    }
24
25    if (0.0) {
26      printf("The double 0.0 is false!\n");
27    }
28
29    return 0;
30  }
```

Booleans

- warning due to narrowing conversion

```
gcc ch03/booleans.c
```

```
ch03/booleans.c:13:7: warning: implicit conversion from 'double' to '_Bool' changes value from 3.14 to true [-Wliteral-conversion]
  if (3.14) {
  ~~  ^~~~
1 warning generated.
```

```
./a.out
```

```
1 is true!
The double 3.14 is true!
The char 'a' is true!
The int -1 is true!
```

- _bool or _Bool introduced, but it's an add-on
- when printing a bool (alias for _bool), 0 or 1 is output

Booleans

- true and false exist as constants

```
cat -n ch03/bool.c
```

```
     1  #include <stdbool.h>
     2  #include <stdio.h>
     3
     4  int main(int argc, char **argv)
     5  {
     6    bool T = true;
     7    bool F = false;
     8
     9    printf("T = %d\n", T);
    10    printf("F = %d\n", F);
    11
    12    if (T == 1) {
    13      printf("T == 1\n");
    14    }
    15
    16    if (F == 0) {
    17      printf("F == 0\n");
    18    }
    19
    20    return 0;
    21  }
```

```
gcc ch03/bool.c
```

```
./a.out
```

```
T = 1
F = 0
T == 1
F == 0
```

Boolean Operators

- logical AND: &&
- logical OR: ||
- logical NOT: !
- be sure to use && and ||, as & and | are bitwise operators

Boolean Operators

```
cat -n ch03/and_or_not.c
```

```
     1  #include <stdio.h>
     2
     3  int main(void)
     4  {
     5    printf("%d\n", (42 < 54) && (42 > 9));
     6    printf("%d\n", (42 < 9) || (42 < 54));
     7    printf("%d\n", !(42 < 54));
     8
     9    return 0;
    10  }
```

```
gcc ch03/and_or_not.c
```

```
./a.out
```

```
1
1
0
```

Comparison Operators

- can't use a <= x <= b in C
  - interpreted as (a <= x) <= b, where the first part returns 0 or 1

```
cat -n ch03/bool_exp.c
```

```
 1  #include <stdio.h>
 2
 3  int main(int argc, char **argv)
 4  {
 5    float x = 42;
 6
 7    printf("x = %.1f\n", x);
 8    if (0 <= x <= 1) {
 9      printf("whoa!  0 <= %.1f <= 1 evaluates to true!\n", x);
10    }
11
12    x = -1.0;
13    printf("x = %.1f\n", x);
14    if ((0 <= x) <= 0.5) {
15      printf("ok!  (0 <= %.1f) <= 0.5 evaluates to true!\n", x);
16    }
17
18    return 0;
19  }
```

Comparison Operators

- may get a warning at compilation

```
clang -pedantic -Wall ch03/bool_exp.c
```

```
ch03/bool_exp.c:8:14: warning: result of comparison of constant 1 with boolean expression is always true [-Wtautological-constant-compare]
  if (0 <= x <= 1) {
      ~~~~~~ ^  ~
1 warning generated.
```

- output

```
./a.out
```

```
x = 42.0
whoa!  0 <= 42.0 <= 1 evaluates to true!
x = -1.0
ok!  (0 <= -1.0) <= 0.5 evaluates to true!
```

Comparison Operators

- Python version

```
cat -n ch03/bool_exp.py
```

```
 1  x = 42.0
 2  print(f"{x = }")
 3  if 0 <= x <= 1:
 4      print(f"whoa!  0 <= {x} <= 1 evaluates to True!")
 5  else:
 6      print(f"yay!  0 <= {x} <= 1 evaluates to False!")
 7
 8  x = -1
 9  print(f"{x = }")
10  if (0 <= x) <= 1:
11      print(f"whoa!  (0 <= {x}) <= 1 evaluates to True!")
12  else:
13      print(f"yay!  0 <= {x} <= 1 evaluates to False!")
```

```
python ch03/bool_exp.py
```

```
x = 42.0
yay!  0 <= 42.0 <= 1 evaluates to False!
x = -1
whoa!  (0 <= -1) <= 1 evaluates to True!
```

Assignment Expressions

- be careful with == and =

```
cat -n ch03/assign_exp.c
```

```
     1  #include <stdio.h>
     2
     3  int main(int argc, char **argv)
     4  {
     5    float x = 42;
     6
     7    printf("x = %.1f\n", x);
     8    if (x = 54) {
     9      printf("yikes!\n");
    10    }
    11
    12    return 0;
    13  }
```

```
clang -pedantic -Wall ch03/assign_exp.c
```

```
ch03/assign_exp.c:8:9: warning: using the result of an assignment as a condition without parentheses [-Wparentheses]
  if (x = 54) {
      ~~^~~~
ch03/assign_exp.c:8:9: note: place parentheses around the assignment to silence this warning
  if (x = 54) {
        ^
      (     )
ch03/assign_exp.c:8:9: note: use '==' to turn this assignment into an equality comparison
  if (x = 54) {
        ^
        ==
1 warning generated.
```

```
./a.out
```

```
x = 42.0
yikes!
```

Assignment Expressions

- can be useful in some instances

```
cat -n ch03/assign_exp4.c
```

```
 1  #include <stdio.h>
 2
 3  int main(int argc, char **argv)
 4  {
 5    int c;
 6    FILE *fp;
 7    char filename[] = "muskmelon.txt";
 8
 9    if ((fp = fopen(filename, "r")) != NULL) {
10      while ((c = getc(fp)) != EOF) {
11          putchar(c);
12      }
13      fclose(fp);
14    }
15    else {
16      printf("File \"%s\" not found!\n", filename);
17    }
18
19    return 0;
20  }
```

Numerical Value of Boolean Expression

- recall short circuiting
- C/C++ always returns 0 or 1 for boolean expressions, but Python may return a different result
- Python return the first value it encounters that allows it to determine the truth of an expression

```
n = (1 >= 2) or (3.14)
```

- Python: n = 3.14 (first value that determines truth of expr)
- C/C++: n = 1

Conditional Expressions

- also called ternary expressions

```
m = (a > b) ? a : b;
```

- above is equivalent to

```
if (a > b) {
m = a;
}
else {
m = b;
}
```

- or the Python statement

```
m = a if a > b else b
```

- place Boolean expression in parentheses for clarity (K&R)

Conditional Expressions

- example used for assigning max of two numbers

```
cat -n ch03/max.c
```

```
     1  #include <stdio.h>
     2
     3  int main(int argc, char **argv)
     4  {
     5    int a = 42;
     6    int b = 54;
     7    int m;
     8
     9    m = (a > b) ? a : b;
    10    printf("max(%d, %d) = %d\n", a, b, m);
    11
    12    return 0;
    13  }
```

```
./a.out
```

```
max(42, 54) = 54
```

Conditional Expressions

- example used for handling the singular/plural of hours

```
cat -n ch03/cond_expr.c
```

```
     1  #include <stdio.h>
     2
     3  int main(void)
     4  {
     5    int duration;
     6
     7    duration = 1;
     8    printf("The duration is %d hour%s.\n", duration, (duration > 1) ? "s" : "");
     9
    10    duration = 2;
    11    printf("The duration is %d hour%s.\n", duration, (duration > 1) ? "s" : "");
    12
    13    return 0;
    14  }
```

```
gcc ch03/cond_expr.c
```

```
./a.out
```

```
The duration is 1 hour.
The duration is 2 hours.
```

Statements and Blocks

- { } to delimit blocks (e.g., if body, while body, etc.)
- indentation irrelevant, though customary for blocks

```
cat -n ch03/scope.c
```

```
     1  #include <stdio.h>
     2
     3  int main()
     4  {
     5    int i = 0;
     6    printf("just before the loop, i = %d\n", i);
     7
     8    for (int i = 1; i <= 5; i++) {
     9      printf("inside the loop, i = %d\n", i);
    10    }
    11
    12    printf("just after the loop, i = %d\n", i);
    13  }
```

```
gcc ch03/scope.c
```

```
./a.out
```

```
just before the loop, i = 0
inside the loop, i = 1
inside the loop, i = 2
inside the loop, i = 3
inside the loop, i = 4
inside the loop, i = 5
just after the loop, i = 0
```

Statements and Blocks

- errors for accessing variables outside of scope

```
cat -n ch03/scope2.c
```

```
     1  #include <stdio.h>
     2
     3  int main()
     4  {
     5    for (int i = 1; i <= 5; i++) {  /* i is local to for statement. */
     6      printf("inside the loop, i = %d\n", i);
     7    }
     8    printf("just after the loop, i = %d\n", i);
     9
    10    {
    11      int j = 42;                   /* j is local to this block. */
    12    }
    13    printf("just after the block, j = %d\n", j);
    14
    15    int k = 54;                     /* k is available at any later point. */
    16    printf("at the end of the program, k = %d\n", k);
    17  }
```

```
gcc ch03/scope2.c
```

```
ch03/scope2.c:8:43: error: use of undeclared identifier 'i'
  printf("just after the loop, i = %d\n", i);
                                          ^
ch03/scope2.c:13:44: error: use of undeclared identifier 'j'
  printf("just after the block, j = %d\n", j);
                                           ^
2 errors generated.
```

if Statement

- **similar to Python, except**
  - parentheses around condition
  - else if instead of elif
- **curly braces may be omitted for body with single statement**
  - similarly for while and for
  - may be clearer to use them anyway

if Statement

- example

```
cat -n ch03/if.c
```

```
 1  #include <stdio.h>
 2
 3  int main(int argc, char **argv)
 4  {
 5    int n = 42;
 6    int m = 54;
 7
 8    /* Use {} even for one-line blocks. */
 9    if (n > 42) {
10      printf("42\n");
11    }
12    else if (m > 42) {
13      printf("54\n");
14    }
15    else {
16      printf("boo!\n");
17    }
18
19    /* Legal, but not advisable. */
20    if (n > 42)
21      printf("42\n");
22    else if (m > 42)
23      printf("54\n");
24    else
25      printf("boo!\n");
26
27    n =  9 * 6;
28    if (n > 42) printf("n: %d\n", n);
29
30    return 0;
31  }
```

```
gcc ch03/if.c
```

```
./a.out
```

```
54
54
n: 54
```

while Statement

- example

```
cat -n ch03/while.c
```

```
     1  #include <stdio.h>
     2
     3  int main(int argc, char **argv)
     4  {
     5    int n = 42;
     6
     7    while (n < 54) {
     8      printf("%d\n", n);
     9      n++;  /* Increment n by 1. */
    10    }
    11
    12    return 0;
    13  }
```

```
gcc ch03/while.c
```

```
./a.out
```

```
42
43
44
45
46
47
48
49
50
51
52
53
```

Increment/Decrement Operators

- **++ and –**
- **behavior depends on whether they are prefix or postfix operators**
  - **`++n`** (prefix increment) means increase **`n`** by **`1`** and use the resulting value
  - **`n++`** (postfix increment) means use the value of **`n`** and then increase by **`1`**
  - **`--n`** and **`n--`** behave similarly

Increment/Decrement Operators

- ++ and --

```
cat -n ch03/increment.c
```

```
 1  #include <stdio.h>
 2
 3  int main(int argc, char **argv)
 4  {
 5    int n = 42;
 6
 7    printf("n: %d\n", n);
 8    n++;  // n = n + 1
 9    printf("n: %d\n", n);
10    n--;  // n = n - 1
11    printf("n: %d\n\n", n);
12
13    printf("Prefix ++ and --\n");
14    n = 42;
15    printf("n at start: %d\n", n);
16    printf("++n: %d\n", ++n);  // Set n = n + 1 and print n.
17    printf("++n: %d\n", ++n);  // Set n = n + 1 and print n.
18    printf("--n: %d\n", --n);  // Set n = n - 1 and print n.
19    printf("--n: %d\n\n", --n);  // Set n = n - 1 and print n.
20
21    printf("Suffix ++ and --\n");
22    n = 42;
23    printf("n at start: %d\n", n);
24    printf("n++: %d\n", n++);  // Print n and set n = n + 1.
25    printf("n++: %d\n", n++);  // Print n and set n = n + 1.
26    printf("n--: %d\n", n--);  // Print n and set n = n - 1.
27    printf("n--: %d\n", n--);  // Print n and set n = n - 1.
28    printf("n at end: %d\n", n);
29
30    return 0;
31  }
```

```
gcc ch03/increment.c
```

```
./a.out
```

```
n: 42
n: 43
n: 42

Prefix ++ and --
n at start: 42
++n: 43
++n: 44
--n: 43
--n: 42

Suffix ++ and --
n at start: 42
n++: 42
n++: 43
n--: 44
n--: 43
n at end: 42
```

for Statement

- **more general than for in Python**
  - equivalent to while statement
- **no iterable objects in C**

for Statement

- for and while statements

```
cat -n ch03/for.c
```

```
 1  #include <stdio.h>
 2
 3  int main(void)
 4  {
 5    int i;
 6
 7    /* A for loop. */
 8    for (i = 42; i < 54; i++) {
 9      printf("%d ", i);
10    }
11    printf("\n");
12
13    /* An equivalent while loop. */
14    i = 42;
15    while (i < 54) {
16          printf("%d ", i);
17          i++;
18    }
19    printf("\n");
20
21    return 0;
22  }
```

```
gcc ch03/for.c
```

```
./a.out
```

```
42 43 44 45 46 47 48 49 50 51 52 53
42 43 44 45 46 47 48 49 50 51 52 53
```

```
cat -n ch03/for2.c
```

```
 1  #include <stdio.h>
 2
 3  int main(void)
 4  {
 5    int i;
 6
 7    for (i = 42; i < 54; i++) {  /* Suffix ++. */
 8      printf("%d ", i);
 9    }
10    printf("\n");
11
12    for (i = 42; i < 54; ++i) {  /* Prefix ++. */
13      printf("%d ", i);
14    }
15    printf("\n");
16
17    return 0;
18  }
```

```
gcc ch03/for.c
```

```
./a.out
```

```
42 43 44 45 46 47 48 49 50 51 52 53
42 43 44 45 46 47 48 49 50 51 52 53
```

for Statement

- common to have temporary local loop variable

```
cat -n ch03/for3.c
```

```
     1  #include <stdio.h>
     2
     3  int main()
     4  {
     5    int i = 0;
     6    printf("Just before the loop, i = %d.\n", i);
     7
     8    for (int i = 1; i <= 5; i++)  {  /* The i declared here is local to the loop. */
     9      printf("Inside the loop, i = %d.\n", i);
    10    }
    11
    12    printf("Just after the loop, i = %d.\n", i);
    13  }
```

```
gcc ch03/for3.c
```

```
./a.out
```

```
Just before the loop, i = 0.
Inside the loop, i = 1.
Inside the loop, i = 2.
Inside the loop, i = 3.
Inside the loop, i = 4.
Inside the loop, i = 5.
Just after the loop, i = 0.
```

do while Statement

- condition checked at end of loop
- guaranteed to be executed at least once

```
cat -n ch03/do.c
```

```
     1  #include <stdio.h>
     2
     3  int main(int argc, char **argv)
     4  {
     5    int n = 42;
     6
     7    while (n < 42) {  /* Test at top of loop; loop does not execute. */
     8      printf("Hello from the while loop!\n");
     9    }
    10
    11    do {              /* Test at bottom of loop; loop executes once. */
    12      printf("Hello from the do loop!\n");
    13    } while (n < 42);
    14
    15    return 0;
    16  }
```

```
gcc ch03/do.c
```

```
./a.out
```

```
Hello from the do loop!
```

break Statement

- terminates execution of the smallest enclosing loop or switch

```
cat -n ch03/break.c
```

```
 1  #include <stdio.h>
 2
 3  int main(void)
 4  {
 5    int i;
 6
 7    i = 0;
 8    while (i < 10) {
 9      if (i == 3) {
10        printf("Skipping 3!\n");
11        i++;
12        continue;  /* Jump to the start of the while loop. */
13      }
14      printf("i: %d\n", i);
15      i++;
16    }
17    printf("\n");
18
19    for (int i = 0; i < 3; i++) {
20      for (int j = 0; j < 4; j++) {
21        if (j == 2) {
22          printf("Skipping j = 2.\n");
23          continue;  /* Jump to the start of the inner for loop. */
24        }
25        printf("i: %d  j: %d\n", i, j);
26      }
27    }
28
29    return 0;
30  }
```

```
gcc ch03/break.c
```

```
./a.out
```

```
i: 0
i: 1
i: 2
Skipping 3!
i: 4
i: 5
i: 6
i: 7
i: 8
i: 9

i: 0  j: 0
i: 0  j: 1
Skipping j = 2.
i: 0  j: 3
i: 1  j: 0
i: 1  j: 1
Skipping j = 2.
i: 1  j: 3
i: 2  j: 0
i: 2  j: 1
Skipping j = 2.
i: 2  j: 3
```

continue Statement

- terminates execution of the smallest enclosing loop

```
cat -n ch03/continue.c
```

```
 1  #include <stdio.h>
 2
 3  int main(void)
 4  {
 5    int i;
 6
 7    i = 0;
 8    while (i < 10) {
 9      if (i == 3) {
10        printf("Skipping 3!\n");
11        i++;
12        break;  /* Jump out of the while loop. */
13      }
14      printf("i: %d\n", i);
15      i++;
16    }
17    printf("\n");
18
19    for (int i = 0; i < 3; i++) {
20      for (int j = 0; j < 4; j++) {
21        if (j == 2) {
22          printf("Skipping j = 2.\n");
23          break;  /* Terminate the inner for loop. */
24        }
25        printf("i: %d  j: %d\n", i, j);
26      }
27    }
28
29    return 0;
30  }
```

```
gcc ch03/continue.c
```

```
./a.out
```

```
i: 0
i: 1
i: 2
Skipping 3!

i: 0  j: 0
i: 0  j: 1
Skipping j = 2.
i: 1  j: 0
i: 1  j: 1
Skipping j = 2.
i: 2  j: 0
i: 2  j: 1
Skipping j = 2.
```

switch Statement

- **no analog in Python**
- **multiway decision statement**
- **checks value against constant int expressions and branches accordingly**
- **executed as follows**
  - control expression is evaluated
  - if value matches a case label, the program jumps to the case block
  - if the value is not a match for case label, the default block is executed
  - if no default case, no statements will be executed

switch Statement

```
cat -n ch03/switch.c
```

```
 1  #include <stdio.h>
 2
 3  int main(void)
 4  {
 5    int n;
 6
 7    n = 2;
 8    switch (n) {
 9      case 1: printf("* ");
10      case 2: printf("** ");    /* Jump to here. */
11      case 3: printf("*** ");   /* This statement will also be executed. */
12      case 4: printf("**** ");  /* As will this one. */
13    }
14    printf("\nDone with first switch statement!\n\n");
15
16    n = 7;
17    switch (n) {
18      case 1: printf("* ");
19      case 2: printf("** ");
20      case 3: printf("*** ");
21      case 4: printf("**** ");
22    }
23    printf("\nDone with second switch statement!\n\n");
24
25    n = 7;
26    switch (n) {
27      case 1: printf("* ");
28      case 2: printf("** ");
29      case 3: printf("*** ");
30      case 4: printf("**** ");
31      default: printf("No stars for you!");
32    }
33    printf("\nDone with third switch statement!\n\n");
34
35    n = 2;
36    switch (n) {
37      case 2: printf("** ");
38      case 1: printf("* ");
39      case 3: printf("*** ");
40      case 4: printf("**** ");
41    }
42    printf("\nDone with fourth switch statement!\n\n");
43
44    return 0;
45  }
```

```
gcc ch03/switch.c
```

```
./a.out
```

```
** *** ****
Done with first switch statement!


Done with second switch statement!

No stars for you!
Done with third switch statement!

** * *** ****
Done with fourth switch statement!
```

switch Statement

- all cases executed due to lack of break statements between cases
- corrected:

```
cat -n ch03/switch2.c
```

```
     1  #include <stdio.h>
     2
     3  int main(void)
     4  {
     5    int n;
     6
     7    n = 2;
     8    switch (n) {
     9      case 1: printf("* ");
    10              break;
    11      case 2: printf("** ");
    12              break;
    13      case 3: printf("*** ");
    14              break;
    15      case 4: printf("**** ");
    16              break;
    17    }
    18
    19    return 0;
    20  }
```

```
gcc ch03/switch2.c
```

```
./a.out
```

```
**
```

switch Statement

- can use fall through behavior to our advantage

```
cat -n ch03/switch3.c
```

```
     1  #include <stdio.h>
     2
     3  int main(void)
     4  {
     5    char c;
     6
     7    c = '2';
     8    switch (c) {
     9      case '0': case '1': case '2': case '3': case '4':
    10      case '5': case '6': case '7': case '8': case '9':
    11          printf("c is a digit!\n");
    12          break;
    13      case ' ': case '\n': case '\t':
    14          printf("c is white space!\n");
    15          break;
    16      default:
    17          printf("c is something else!\n");
    18    }
    19
    20    return 0;
    21  }
```

```
gcc ch03/switch3.c
```

```
./a.out
```

```
c is a digit!
```