Chapter 13
Control Structures

Control Structures
Conditional
• Making decision about which code to execute, based on evaluated expression
  • if
  • if-else
  • switch

Iteration
• Executing code multiple times, ending based on evaluated expression
  • while
  • for
  • do-while

If
if (condition)
  action;

Condition is a C expression, which evaluates to TRUE (non-zero) or FALSE (zero).
Action is a C statement, which may be simple or compound (a block).

Example If Statements
if (x <= 10)
  y = x * x + 5;

if (x <= 10) {
  y = x * x + 5;
  z = (2 * y) / 3;
}

if (x <= 10)
  y = x * x + 5;
  z = (2 * y) / 3;

Style: avoid singleton if statements (I really dislike them)
compound statement; both executed if x <= 10
only first statement is conditional; second statement is always executed
More If Examples

```c
if (0 <= age && age <= 11) {
    kids = kids + 1;
}
if (month == 4 || month == 6 ||
    month == 9 || month == 11) {
    printf("The month has 30 days.\n");
}
if (x = 2) {
    y = 5;
}
```

This is a common programming error (= instead of ==), not caught by compiler because it's syntactically correct.

Generating Code for If Statement

```c
if (x == 2) {
    y = 5;
}
```

```c
LDR R0, R6, #0 ; load x into R0
ADD R0, R0, #-2 ; subtract 2
BRnp NOT_TRUE ; if non-zero, x is not 2
AND R1, R1, #0 ; store 5 to y
ADD R1, R1, #5
STR R1, R6, #1
NOT_TRUE ...
; next statement
```

Generating Code for If-Else

```c
if (x) {
    y++; z--;
} else {
    y--; z++;
}
```

```c
LDR R0, R6, #0; x is not zero
BRz ELSE
LDR R1, R6, #1; incr y
ADD R1, R1, #1
STR R1, R6, #1
ADD R1, R1, #1
AND R1, R1, #0
ADD R1, R1, #5
STR R1, R6, #1
JMP DONE; skip else code
ELSE LDR R1, R6, #1; decr y
ADD R1, R1, #1
STR R1, R6, #1
ADD R1, R1, #1
STR R1, R6, #1
DONE ...; next statement
```

If-else

```
if (condition)
    action_if;
else
    action_else;
```

Else allows choice between two mutually exclusive actions without re-testing condition.
Matching Else with If

Else is always associated with closest unassociated if

```java
if (x != 10) {
    if (y > 3)
        z = z / 2;
    else
        z = z * 2;
}

is the same as...

```java
if (x != 10) {
    if (y > 3)
        z = z / 2;
    else
        z = z * 2;
}
```

is NOT the same as...

```java
if (x != 10) {
    if (y > 3)
        z = z / 2;
} else
    z = z * 2;
```

Solution: always use braces (avoids the problem entirely)

Chaining If’s and Else’s

```java
if (month == 4 || month == 6 || month == 9 ||
    month == 11) {
    printf("Month has 30 days.

    Other days: \n");
} else if (month == 1 || month == 3 ||
    month == 5 || month == 7 ||
    month == 8 || month == 10 ||
    month == 12) {
    printf("Month has 31 days.

    Other days: \n");
} else if (month == 2) {
    printf("Month has 28 or 29 days.

    Other days: \n");
} else {
    printf("Don’t know that month.

    Other days: \n");
}
```

Generating Code for While

```assembly
x = 0;
while (x < 10) {
    printf("%d", x);
    x = x + 1;
}
```

```assembly
AND R0, R0, #0
STR R0, R6, #0 ;x = 0
:test
LDR R0, R6, #0 ;load x
ADD R0, R0, #-10
BRzp DONE ;loop body
LDR R0, R6, #0 ;load x
...
<printf>
ADD R0, R0, #1 ;incr x
STR R0, R6, #0
JMP LOOP ;test again
DONE ;next statement
```
**Infinite Loops**

The following loop will never terminate:

```c
x = 0;
while (x < 10) {
    printf("%d ", x);
}
```

Loop body does not change condition...
- ...so test is never false

Common programming error that can be difficult to find.

---

**Generating Code for For**

```c
for (i = 0; i < 10; i++) {
    printf("%d ", i);
}
```

This is the same as the while example!

---

**For**

```c
for (init; end-test; re-init)
    statement
```

Executes loop body as long as test evaluates to TRUE (non-zero). Initialization and re-initialization code included in loop statement.

*Note: Test is evaluated **before** executing loop body.*

---

**Example For Loops**

```c
/* -- what is the output of this loop? -- */
for (i = 0; i <= 10; i++) {
    printf("%d ", i);
}
```

```c
/* -- what does this one output? -- */
for (c = 0; c < 26; c++) {
    printf("%c ", letter+c);
}
```

```c
/* -- what does this loop do? -- */
numberOfOnes = 0;
for (bitNum = 0; bitNum < 16; bitNum++) {
    if (inputValue & (1 << bitNum)) {
        numberOfOnes++;
    }
}
```
Nested Loops

Loop body can (of course) be another loop

```c
/* print a multiplication table */
for (mp1 = 0; mp1 < 10; mp1++) {
    for (mp2 = 0; mp2 < 10; mp2++) {
        printf("%d\t", mp1*mp2);
    }
    printf("\n");
}
```

Another Nested Loop

Here, test for the inner loop depends on counter variable of outer loop

```c
for (outer = 1; outer <= input; outer++) {
    for (inner = 0; inner < outer; inner++) {
        sum += inner;
    }
}
```

For vs. While

In general:

**For** loop is preferred for counter-based loops
- Explicit counter variable
- Easy to see how counter is modified each loop

**While** loop is preferred for sentinel-based loops
- Test checks for sentinel value.

Either kind of loop can be expressed as other, so really a matter of style and readability

Do-While

```c
do
    loop_body;
while (test);
```

Executes loop body as long as test evaluates to TRUE (non-zero).

*Note: Test is evaluated **after** executing loop body*
Break and Continue

`break;`
- used only in switch statement or iteration statement
- passes control out of the “smallest” (loop or switch) statement containing it to the statement immediately following
- usually used to exit a loop before terminating condition occurs (or to exit switch statement when case is done)

`continue;`
- used only in iteration statement
- terminates the execution of the loop body for this iteration
- loop expression is evaluated to see whether another iteration should be performed
- if for loop, also executes the re-initializer

Example

What does the following loop do?
```c
for (i = 0; i <= 20; i++) {
    if (i%2 == 0) {
        continue;
    }
    printf("%d ", i);
}
```

What would be an easier way to write this?

What happens if `break` instead of `continue`?

Switch

```c
switch (expression) {
    case const1:
        action1;
        break;
    case const2:
        action2;
        break;
    default:
        action3;
}
```

Alternative to long if-else chain.
If break is not used, then case “falls through” to the next.

Switch Example

```c
/* same as month example for if-else */
switch (month) {
    case 4:
    case 6:
    case 9:
    case 11:
        printf("Month has 30 days.\n");
        break;
    case 1:
    case 3:
        /* some cases omitted for brevity...*/
        printf("Month has 31 days.\n");
        break;
    case 2:
        printf("Month has 28 or 29 days.\n");
        break;
    default:
        printf("Don’t know that month.\n");
}
```
More About Switch

Case expressions must be constant

```c
case i: /* illegal if i is a variable */
```

If no break, then next case is also executed

```c
switch (a) {
  case 1:
    printf("A");
    /* a is 1, prints “ABC”.
    If a is 2, prints “BC”.
    Otherwise, prints “C”.
  
  case 2:
    printf("B");
    
  default:
    printf("C");
}
```

Aside: Enumerations

Keyword `enum` declares a new type

- `enum colors { RED, GREEN, BLUE, GREEN, YELLOW, MAUVE };`
- RED is now 0, GREEN is 1, etc.
- Gives meaning to constants, groups constants

```c
eenum colors house_color;
house_color = get_color();
switch (house_color) {
  case RED:
    /* code here */
    break;
/* more here... */
}
```

 Enums are just ints, but can provide more type checking
- Warning on assignment (example: `house_color = 85;`)
- Warning on “partial” switch statement
- C++ adds even more checking support

Example: Searching for Substring

Have user type in a line of text (ending with linefeed) and print the number of occurrences of "the"

Reading characters one at a time
- Use the `getchar()` function -- returns a single character

Don’t need to store input string; look for substring as characters are being typed
- Similar to state machine: based on characters seen, move toward success state or move back to start state
- Switch statement is a good match to state machine

Substring: State machine to flow chart

```
read char
```

```
if 't', match=1
```

```
match = 0
```

```
T
```

```
match = 1
```

```
if 't', match=1
```

```
F
```

```
if 'h', match=2
```

```
T
```

```
match = 2
```

```
if 'e', count++ and match = 0
```

```
T
```

```
if 't', match=1
```

```
else match=0
```

```
F
```

```
F
```

```
else match=0
```

```
F
```

```
F
```
#include <stdio.h>
enum state { NO_MATCH, ONE_MATCH, TWO_MATCHES };
main()
{
    char key; /* input character from user */
    int match = NO_MATCH; /* state of matching */
    int count = 0; /* number of substring matches */
    /* Read character until newline is typed */
    key = getchar();
    while (key != 'n') {
        /* Action depends on number of matches so far */
        switch (match) {
        }
        key = getchar();
    }
    printf("Number of matches = %d\n", count);
}

Substring: Code (Part 2)
    case NO_MATCH: /* starting - no matches yet */
        if (key == 't') {  
            match = ONE_MATCH;
        } else {  
            match = NO_MATCH;
        }  
        break;
    case ONE_MATCH: /* 't' has been matched */
        if (key == 'h') {  
            match = TWO_MATCHES;
        } else if (key == 't') {  
            match = ONE_MATCH;
        } else {  
            match = NO_MATCH;
        }  
        break;
    case TWO_MATCHES: /* 'th' has been matched */
        if (key == 'e') {  
            count++; /* increment count */
            match = NO_MATCH; /* go to starting point */
        } else if (key == 't') {  
            match = ONE_MATCH;
        } else {  
            match = NO_MATCH;
        }  
        break;