Control Flow

Control-flow statements specify the order in which computations are performed.

This lecture deals with this topic (chapter 3 K&R)

Expressions and Side Effects

(i = 1) + (j = 2)

1 + 2

Both are expressions in C

Both yield the value 2

What is the difference?

The first expression has side effect (i gets 1 and j gets 2)

The second has no side effects

Expressions and Side Effects

In C all operators yield values.

Assignment is an operator, so it produces a value.

An operator may have side effects

i = 1 yields the value 1 and has side effect (x gets 1)

value may be discarded:
e.g., i = 1;

value may be used:
e.g., i = j = 1

Expression vs Statement

(Rule) Any expression can become a statement by appending a semicolon to it.

E.g. i = 1 vs i = 1;

i = 1 is an expression (with value 1) that has the side effect of assigning 1 to i whenever the expression is evaluated

i = 1; is an assignment. (the value is discarded here)

1 + 2 ; legal statement (does nothing when executed)

i++ ; i--; are statements (side effect: changes i)

statements with side effects are useful.

Blocks

The semicolon in C is a statement terminator rather than a statement separator (Pascal)

Braces { and } are used to group declarations and statements together in a compound statement or a block, so that they are syntatically equivalent to a single statement.

A compound statement can appear wherever a single statement can appear (general rule)

eg. { i = 0; j = i;}

Notice that there is no semicolon after the right brace that ends a block.

Variables can be declared inside any block.

If-Else

```
if    (expression)
      statement_1
else
      statement_2
```

else part is optional.

if expression is true statement_1 is executed.
if expression is false statement_2 is executed.

true corresponds the expression has non-zero value.
false corresponds the expression has zero value.

if the else part is omitted, control goes to the next statement
when the expression is false.

If-Else usage

if (expression) vs if (expression != 0)

```
if (n > 0)
      if (a > b)
            z = a;
      else
            z = b;
```

The else goes with the inner if.

```
if (n > 0) {
      if (a > b)
            z = a;
}
else
      z = b;
```

The else goes with the outer if. Notice the use of braces

Else-If

```
if     (expression)
       statement
else  if  (expression)
       statement
else  if  (expression)
       statement
else
       statement
```

The expressions are evaluated in order, if any expression is true the corresponding statement is executed and this terminates the whole chain.
Any number of else if can be in the chain. the final else statement is the default case

This construct is most general way of writing a multiway decision.

Switch

switch is a special case of multiway decision.

```
switch  (expression)  {
        case   const-expr:  statements
        case   const-expr:  statements
        default:   statements
}
```

The const-expr must be a constant that has the char type, int type or enumerated type.

the default case is executed if none of the other cases is satisfied. deault is optional.

Switch - example (K & R page 59)

```
/* count digits,  white space and others  */
...
while ((c = getchar()) != EOF) {
        switch(c) {
        case '0':  case '1':  case '2':  case '3':  case '4':
        case '5':  case '6':  case '7':  case '8':  case '9':
                ndigit[c - '0']++;
                break;
        case ' ':
        case '\n':
        case '\t':
                nwhite++;
                break;
        default:
                nother++;
                break;
        }
}
```

Loops - While and For

```
while  (expression)
      statement


for  (expr_1;    expr_2;    expr_3)
      statement
```

is equivalent to

```
expr_1;
while  (expr_2)  {
      statement
      expr_3;
}
```

For loop

```
for  (expr_1;    expr_2;    expr_3)
       statement
```

expr_1 and expr_3 are often assignments or function calls and
expr_2 is a relational expression.

Any of the three parts can be omitted, but semicolons must remain.

if expr_2 is not present it is taken as permanently true.
This is the famous forever loop (infinite loop)

```
       for  (; ;)   {
              ...
       }
```

The loop is often broken by using a break or return.

```
for   (i  = 0;    i  <  n;  i++)
       a [i]  =  0;              /*   is a C idiom  */
```

Comma operator

The comma operator is often used with for statement.

A pair of expressions separated by a comma is evaluated left to right , and the type and value of the result are the type and the value of the right operand.

thus in a for loop , it is possible to place multiple expressions in various parts, for example to process two indices in parallel.

E.g.,

```
for ( i = 0, j = 100; i != 10; ++i, j -= 10 )
    ....
```

initializes the value of i to 0 and j to 100 before the loop begins, and increments the value of i and subtracts 10 from the value of j each time after the body of the loop is executed.

Comma operator

A comma used to separate the arguments in a function call , or variable names in a list of declarations is a separate syntactic entity and is not an instance of comma operator.

```
/* reversal of an n element array*/

 for ( i =  0, j = n-1;  i < j;  i++, j--) {

        temp  = a[i];
        a[i]  =  a[j];
        a[j]  =  temp;
}
```

Do while

```
do
      statement
while (expr)
```

/* the loop counts input characters excluding spaces and line-feeds*/

```
do {
        c  =  getchar();
        if   (c  !=  ' '  &&   c  !=  '\n')
             ccount++;
}  while  (c  !=  EOF);
```

Break and Continue

break statement is used to exit a loop from within a loop body, control transferred to the statement immediately after the loop.

Only one level of loop nesting can be exited with break.

continue has the opposite effect.

Inside while / do while it causes control to be transferred to the top of the loop.

Inside a for loop control passes to the last expression in the loop control, expr_3.

Break and Continue - Example

```
/* counts lines and non-blank characters  */

for (;  ;)    {
        int    c;
        c  =  getchar();
        if  ( c  ==' '  ||    c  ==  '\t')
                continue;
        if  ( c  ==' \n' )   {
                line_count++;
                continue;
        }
        if   (c  == EOF)   break;
                ccount++;
}
```

Goto and Labels

A label has a same form as a variable name and it is followed by a colon. It can be attached to any statement in the same function as the goto.The scope of the label is the entire function.

The statement

```
goto label ;
```

transfers control to the statement label.

Good programs seldom use goto.