Flow of Control

Flow of Control

- C is a sequential language
  - statements in a program are executed one after another

- To change flow of control, use
  - choice instructions: if, switch
  - iterative instructions: while, for, fo
  - OR recursion
  - you may need operators

Operators for them

type of operators	informal description	operator mnemonic
relational operators	less than greater than less than or equal to greater than or equal to	< > <= >=
equality operators	equal to not equal to	== !=
logical operators	(unary) negation logical and logical or	! && ||

Relational Operators

Declarations and initializations		
char c = 'w' int i = 1, j = 2, k = -7 double x = 7e+33, y = 0.001		
Expression	**Equivalent expression**	**Value**
'a' + 1 < c	('a' + 1) < c	1
- i - 5 * j >= k + 1	((- i) - (5 * j)) >= (k + 1)	0
3 < j < 5	(3 < j) < 5	1
x - 3.333 <= x + y	(x - 3.333) <= (x + y)	1
x < x + y	x < (x + y)	0?

Equality Operators

- equality expression ::=

  expression == expression |

  expression != expression

- examples
  - c == 'A'
  - k != 2
  - x + y == 3 * z -7
- common mistakes

  = instead of ==

  = =

  =!

Equality Operators Examples

Declarations and initialisations		
`int i = 1, j = 2, k = 3;`		
Expression	**Equivalent expression**	**Value**
`i == j`	`i == j`	0
`i != j`	`i != j`	1
`i + j + k == -2 * -k`	`((i + j) + k) == ((-2) * (-k))`	1

Logical Operators

- logical expressions
  - negative!a
  - or expr || expr
  - and expr && expr
- examples

  !a !(x + 7.3) !(a < b || c < d)

  a && b a ||b !(a<b) && c
- common mistakes

  a!

  a&&

  a & b

  & b --- this is serious

some trivial examples

```
char   c = 'A';
int      i = 7, j = 7;
double   x = 0.0, y = 2.3;
!c                              will return 0
!(i - j)                        will return 1
!i  - j                         will return -7
!! (x + y)                      will return 1
!x * !!y                        will return 1
x || i && j - 3                 will return 1
```

some tricky examples

Declarations and initializations		
char c = 'B' int i = 3, j = 3, k = 3 double x = 0.0, y = 2.3		
Expression	**Equivalent expression**	**Value**
i && j && k	(i && j) && k	1
x || i && j - 3	x || (i && (j - 3))	0
i < j && x < y	(i < j) && (x < y)	0
i < j || x < y	(i < j) || (x < y)	1
'A' <= c && c <= 'Z'	('A' <= c) && (c <= 'Z')	1
c - 1 == 'A' || c + 1 == 'Z'	((c - 1) == 'A') || ((c + 1) == 'Z')	1

short-circuit

- the evaluation stops as soon as the outcome is known
- expr1 && expr2
  - if expr1 is evaluated to be false, expr2 needs not be evaluated
- expr 1 && expr 2

The Compound Statement

- A compound statement is a series of declarations and statements surrounded by braces { }

```
{       int a, b, c;
        a += b += c;
        printf ("a = %d, b = %d, c = %d₩n", a, b, c);
}
```

- a compound is usually called "block"
- expression statements

```
a + b + c;
;  /* empty statement */
```

if statement

- if expr (then) statement | block

```
if (y != 0.0)
   x /= y;
if (c == ' ') {
   ++blank_cnt;
   printf("found another blank\n");
}
if b == a  // parentheses missing
   area = a * a;
```

- statement can be an empty one
- same for else statement

```
if (c >= 'a' && c <= 'z')
    ++lc_cnt;
else{
    ++other_cnt;
    printf("%c is not a lowercase letter\n", c);
}
```

```
if (i != j) {
    i += 1;
    j += 2;
};
else
    i -= j; // syntax error
```

```
if (a == 1)
    if (b == 2)
        printf("***\n");
```

```
if (a == 1)
    if (b == 2)
        printf("***\n");
    else
        printf("###\n");
```

```
if (a == 1)
    if (b == 2)
        printf("***\n");
else
    printf("###\n");
```

The rule:
an *else*
attaches to the
nearest *if*.

Iterative Statements

- while, for, and do statements
  - provide iterative action

- goto, break, continue, return statements cause an unconditional transfer
  - SE people hate these (except return)

```
while (i++ < n)
    factorial *= i;
```

```
while ((c = getchar()) == ' ')
    ; // skip blank characters in the input stream
```

```
while (++i < LIMIT) do {
    // syntax error: do is not allowed
    j = 2 * i + 3;
    printf("%d\n", j);
}
```

```
#include <stdio.h>

int main(void)
{
   int   blank_cnt = 0, c, digit_cnt = 0,
         letter_cnt = 0, nl_cnt = 0, other_cnt = 0;

   while ((c = getchar()) != EOF)      /* braces not necessary */
      if (c == ' ')
         ++blank_cnt;
      else if (c >= '0' && c <= '9')
         ++digit_cnt;
      else if (c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z')
         ++letter_cnt;
      else if (c == '₩n')
         ++nl_cnt;
      else
         ++other_cnt;

   printf("%10s%10s%10s%10s%10s%10s₩n₩n",
      "blanks", "digits", "letters", "lines", "others", "total");
   printf("%10d%10d%10d%10d%10d%10d₩n₩n",
      blank_cnt, digit_cnt, letter_cnt, nl_cnt, other_cnt,
      blank_cnt + digit_cnt + letter_cnt + nl_cnt + other_cnt);
   return 0;
}
```

for statements

- comma operators

```
for (sum = 0, i = 1; i <= n; ++i)
    sum += i;
```

```
for (sum = 0, i = 1; i <= n;  sum += i, ++i)
```

Declarations and initializations		
int i, j, k = 3; double x = 3.3;		
Expression	**Equivalent expression**	**Value**
i = 1, j = 2, ++ k + 1	((i = 1), (j = 2)), ((++ k) + 1)	5
k != 1, ++ x * 2.0 + 1	(k != 1), (((++ x) * 2.0) + 1)	9.6

do statement

- a variant of while statement
  - do { statements } while expr
  - the block is executed first, and then the expr is evaluated
  - you should be able to convert do statement to while statement, and vice versa

```c
/* A test that fails. */

#include <stdio.h>

int main(void)
{
    int       cnt = 0;
    double    sum = 0.0, x;

    for (x = 0.0; x != 9.9; x += 0.1) {    /* trouble! */
        sum += x;
        printf("cnt = %5d\n", ++cnt);
    }
    printf("sum = %f\n", sum);
    return 0;
}
```

nonrobust.c

goto statement

- jump to a label
  - goto label;
  - label: /* label is an identifier */
- it is considered to be harmful, but

```
goto error;
……
error: {
   printf("An error has occurred -bye!\n");
   exit(1);
}
```

```
while (scanf("%lf", &x) == 1){
   if (x<0.0)
        goto negative_alert;
   printf("%f %f\n", sqrt(x), sqrt(2*x));
}

negative_alert: printf("Negative value encountered!\n");
```

break statement

- an exit from a loop

```
while (1) {
    scanf("%lf", &x);
    if (x < 0.0)
        break;
    /* no square root if number is negative, exit loop */
    printf("%f\n", sqrt(x));
}
/* break jumps to here */
```

continue statement

- stop the current iteration and goto the next iteration

```
for (i = 0;  i < TOTAL; ++i) {
      c = getchar();
      if (c >= '0' && c <= '9')
        continue;
      ...              /* processing other characters */
/* continue transfers control to here to begin next iteration
}
```

switch statement

- switch (expr1) /* must be integral */
  - goto the matched case label

```
switch (c) {
case 'a':
      ++a_cnt;
      break;
case 'b':
case 'B':
     ++b_cnt;
     break;
default:
     ++other_cnt;
}
```

conditional operators

x = (y < z) ? y : z

if (y < z) x = y; else x = z;

```
Declarations and initializations

Char    a = 'a', b = 'b';  // a has decimal value 97
int     i = 1, j = 2;
double  x = 7.07;
```

Expression	Equivalent expression	Type	Value
`i == j ? a - 1 : b + 1`	`(i == j) ? (a - 1) : (b + 1)`	int	99
`j % 3 == 0 ? i + 4 : x`	`((j % 3) == 0) ? (i + 4) : x`	double	7.07
`j % 3 ? i + 4 : x`	`(j % 3) ? (i + 4) : x`	double	5.0