KUAS
KYOTO UNIVERSITY OF ADVANCED SCIENCE
京都発世界人財

京都先端科学大学

Information Processing 2 02

the `for` statement
symbolic constants
characters and character input/output
arrays

Ian Piumarta

Faculty of Engineering, KUAS

last week: introduction

course overview
history of C
interpreters vs. compilers
installing the compiler
running the compiler
importance and reading of error messages
importance of writing lots of programs
`hello.c`, compiling, running, `a.out` vs. `-o`
what each line means
`printf` and formatted output
F to C temperature table using `while`
integer and floating types
formatted output

Chapter 1. A Tutorial Introduction
1.1 Getting Started
1.2 Variables and Arithmetic Expressions

this week: character I/O and arrays

1.3 The `for` Statement
1.4 Symbolic Constants
1.5 Character Input and Output
1.6 Arrays

`for` statement and relationship to `while`
symbolic constants `getchar`, `putchar`, simple file copying
character counting
`if` statement
line counting
logical operators `&&` `||`
word counting
arrays, counting occurrences of each digit

`for` statement is shorthand for `while`

very common pattern:

setup (initial conditions, variable values)

```
while ( test ) {
  body
  update
}
```

‘short hand’ notation for the same pattern:

```
for ( setup ; test ; update ) {
  body

}
```

examples:

```
int i;
i = 0;                  // setup
while (i < 10) {        // test
  printf("%d\n", i);    // body
  i = i + 1;            // update
}
```

```
int i;
//   setup    test      update
for (i = 0;  i < 10;  i = i + 1) {
  printf("%d\n", i);   // body
}
```

temperature table using `while`

```c
#include <stdio.h>

int main()
{
    float fahr, celsius;
    int lower, upper, step;

    lower =   0;
    upper = 100;
    step  =  10;

    celsius = lower;                                // SETUP
    while (celsius <= upper) {                      // TEST
        fahr = 32 + celsius * 9 / 5;                // body
        printf("%6.1f %6.1f\n", celsius, fahr);     // body
        celsius = celsius + step;                   // STEP
    }

    return 0;
}
```

→ *replace* `while` *with* `for`

temperature table using `for`

```c
#include <stdio.h>
int main()
{
    float fahr, celsius;
    int lower, upper, step;
    lower =   0;        // lower limit of temperature
    upper = 100;        // upper limit of temperature
    step  =  10;        // step size between temperatures
    //         SETUP             TEST                  STEP
    for (celsius = lower;  celsius <= upper;  celsius = celsius + step) {
        fahr = 32 + celsius * 9 / 5;
        printf("%6.1f %6.1f\n", celsius, fahr);
    }
    return 0;
}
```

symbolic constants

three variables are used only to store constants

```
int lower, upper, step;
lower =   0;        // lower limit of temperature
upper = 100;        // upper limit of temperature
step  =  10;        // step size between temperatures
for (celsius = lower;  celsius <= upper;  celsius = celsius + step) {
    ...
}
```

it would be better to give names to the constants without using variables

symbolic constants

```
#define    name    replacement text
```

in the rest of the program:

- whenever you write *name*
- it will be replaced by *replacement text*

```
//      name      replacement text
#define ZERO           0
#define ONE            1
#define TEN           10
#define FORMAT    "%d\n"

int i;
for (i = ZERO;  i < TEN;  i = i + ONE)
  printf(FORMAT, i);
```

→

```
int i;
for (i = 0;  i < 10;  i = i + 1)
  printf("%d\n", i);
```

symbolic constants

```c
#include <stdio.h>
#define LOWER    0      // lower limit of temperature
int main()
{
    float fahr, celsius;
    int upper, step;
    upper = 100;        // upper limit of temperature
    step  =  10;        // step size between temperatures
    for (celsius = LOWER;  celsius <= upper;  celsius = celsius + step) {
        fahr = 32 + celsius * 9 / 5;
        printf("%6.1f %6.1f\n", celsius, fahr);
    }
    return 0;
}
```

→ *replace* `upper` *and* `step` *with symbolic constants*

symbolic constants

```c
#include <stdio.h>
#define LOWER    0      // lower limit of temperature
#define UPPER  100      // upper limit of temperature
#define STEP    10      // step size between temperatures
int main()
{
    float fahr, celsius;
    celsius = LOWER;
    for (celsius = LOWER;  celsius <= UPPER;  celsius = celsius + STEP) {
        fahr = 32 + celsius * 9 / 5;
        printf("%6.1f %6.1f\n", celsius, fahr);
    }
    return 0;
}
```

`getchar`, `putchar`, copying input to output

characters are integers and should be stored in `int` variables

to read a character from the input (where `c` should be an `int` variable):

```
c = getchar()
```

when there is no more input (e.g., at the end of a file) `getchar()` returns `EOF`

- `EOF` is a symbolic constant indicating “end-of-file”

write a character to the output:

```
putchar(c)
```

combine `getchar` and `putchar` to copy input to output:

read a character from the input, using `getchar`
`while` (*the character is not the end-of-file indicator*)
    write the character to the output, using `putchar`
    read a character from the input, using `getchar`

getchar, putchar, copying input to output

```
#include <stdio.h>      // definitions of getchar, putchar
int main()
{
  int c;                // variable for storing characters
  c = getchar();        // read a character from the input
  while (c != EOF) {    // the character is not end-of-file
    putchar(c);         // write the character to the output
    c = getchar();      // read a character from the input
  }
  return 0;
}
```

assignment is an expression

`c = getchar()` is an *expression*

you can put '`c = getchar()`' inside a larger expression

in particular, you can combine '`c = getchar()`' with '`c != EOF`'

beware: the precedence of = is *lower* than `!=`

- pro tip: *always* use parentheses around an assignment *expression*

```
int c;
while ((c = getchar()) != EOF) {
    putchar(c);
}
```

→ *modify your program to use this pattern*

getchar, putchar, copying input to output

```c
#include <stdio.h>
int main()
{
  int c;
  while ((c = getchar()) != EOF) {
    putchar(c);
  }
  return 0;
}
```

character counting

copy or save your program using a different name (e.g., `char-count.c`)
add another `int` variable to store the number of characters read
initialise it to `0`
add `1` to it each time the loop body runs (don't bother printing the character)
print the value of the variable (number of characters read) at the end

```
let numChars be 0
while ( read next character and it is not the end-of-file indicator )
    add 1 to numChars
print numChars
```

→ *try making these changes*

character counting

```c
#include <stdio.h>
int main()
{
  int c, numChars;
  numChars = 0;
  while ((c = getchar()) != EOF) {
    numChars = numChars + 1;
  }
  printf("%d\n", numChars);
  return 0;
}
```

incrementing variables

incrementing a variable is a common operation:

```
numChars = numChars + N          // increment numChars by N
```

a shorthand form of adding `N` to `numChars` is

```
numChars += N                    // increment numChars by N
```

when `N` is `1` there is an even shorter form

```
++numChars                       // increment numChars by 1
```

→ *use ++ to increment* `numChars` *in your program*

incrementing variables

```c
#include <stdio.h>
int main()
{
  int c, numChars;
  numChars = 0;
  while ((c = getchar()) != EOF) {
    ++numChars;           // count one more character
  }
  printf("%d\n", numChars);
  return 0;
}
```

the `if` statement

`if` uses a condition to control whether a statement is executed

template:

```
if ( expression )
    statement
```

example:

```
if (c == '\n')
    printf("character is newline\n");
```

`else` can immediately follow an `if` statement

it provides another statement to run when the condition is false

```
if (expression)
    statement1
else
    statement2
```

```
if (x < 0)
    printf("x is negative\n");
else
    printf("x is non-negative\n");
```

line counting

```
let numLines be 0
while ( read next character and it is not the end-of-file indicator )
    if ( the character is a newline ) then
        add 1 to numLines          // count the line
print numLines
```

how do we test whether a character is a newline?

- character constants are written using single quotes, e.g., `'x'`
- any single character can appear between the quotes
- including special characters such as *newline*

the newline character is written as a constant like this: `'\n'`

→ *modify your program to count the number of lines*

line counting

```c
#include <stdio.h>
int main()
{
  int c, numLines;
  numLines = 0;
  while ((c = getchar()) != EOF) {
    if (c == '\n')
      ++numLines;           // count number of lines
  }
  printf("%d\n", numLines);
  return 0;
}
```

logical operators: && ||

in C, integer value 0 is “false” and any other value is “true”

`if` (*expression*$_1$ `&&` *expression*$_2$)
⇒ **both** *expression*$_1$ **and** *expression*$_2$

```
0 && 0 == 0
0 && 1 == 0
1 && 0 == 0
1 && 1 == 1
```

`if` (*expression*$_1$ `||` *expression*$_2$)
⇒ **either** *expression*$_1$ **or** *expression*$_2$

```
0 || 0 == 0
0 || 1 == 1
1 || 0 == 1
1 || 1 == 1
```

*if the character is a space **or** a tab **or** a newline then...*

```
if (c == ' ' || c == '\t' || c == '\n') {
    // space or tab or newline
}
else {
    // any other character
}
```

detecting spaces, tabs, newlines

```c
#include <stdio.h>      // printf, getchar, putchar
int main()
{
    int c;
    while ((c= getchar()) != EOF) {                 // not at the end of the file
        if (c == ' ' || c == '\t' || c == '\n')     // character is space, tab, newline
            printf("character is space");
        else
            printf("character is not a space");
    }
    return 0;
}
```

characters are really just integers

characters are integers with standardised values

```
printf("the integer value of '0' is %d\n", '0');
printf("the integer value of '9' is %d\n", '9');
printf("the integer value of 'A' is %d\n", 'A');
printf("the integer value of 'Z' is %d\n", 'Z');
printf("the integer value of 'a' is %d\n", 'a');
printf("the integer value of 'z' is %d\n", 'z');
```

`printf` understands `"%c"` to print a single character

```
putchar(48);  putchar(32);  putchar(65);  putchar(10);  // or...
printf("%c%c%c%c", 48, 32, 65, 10);
```

→ *write a program that prints values of characters from* `' '` *to* `'~'`

standard values of the first 128 characters

the integers that represent characters in programs: ASCII encoding

(American Standard Code for Information Interchange, and also the first 127 Unicode characters)

	0 nul	1 soh	2 stx	3 etx	4 eot	5 enq	6 ack	7 bel
	8 bs	9 ht	10 nl	11 vt	12 np	13 cr	14 so	15 si
	16 dle	17 dc1	18 dc2	19 dc3	20 dc4	21 nak	22 syn	23 etb
	24 can	25 em	26 sub	27 esc	28 fs	29 gs	30 rs	31 us
space →	32 sp	33 !	34 "	35 #	36 $	37 %	38 &	39 ’
	40 (	41)	42 *	43 +	44 ,	45 -	46 .	47 /
	48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7
	56 8	57 9	58 :	59 ;	60 <	61 =	62 >	63 ?
	64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G
	72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
	80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W
	88 X	89 Y	90 Z	91 [	92 \	93]	94 ^	95 _
	96 ‘	97 a	98 b	99 c	100 d	101 e	102 f	103 g
	104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
	112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w
	120 x	121 y	122 z	123 {	124 |	125 }	126 ~	127 del

invisible device control codes

punctuation

digits, punctuation

uppercase letters, punctuation

lowercase letters, punctuation

← delete

counting occurrences of each digit

```c
#include <stdio.h>

int main()
{
    int c;
    int d0, d1, d2, d3, d4, d5, d6, d7, d8, d9;  // ten variables to store digit counts
    d0 = d1 = d2 = d3 = d4 = d5 = d6 = d7 = d8 = d9 = 0;

    while ((c = getchar()) != EOF) {
        if (c == '0') ++d0;
        if (c == '1') ++d1;
        if (c == '2') ++d2;
        if (c == '3') ++d3;
        if (c == '4') ++d4;
        if (c == '5') ++d5;
        if (c == '6') ++d6;
        if (c == '7') ++d7;
        if (c == '8') ++d8;
        if (c == '9') ++d9;
    }
```

continued...

```c
    printf("0 %d\n", d0);
    printf("1 %d\n", d1);
    printf("2 %d\n", d2);
    printf("3 %d\n", d3);
    printf("4 %d\n", d4);
    printf("5 %d\n", d5);
    printf("6 %d\n", d6);
    printf("7 %d\n", d7);
    printf("8 %d\n", d8);
    printf("9 %d\n", d9);

    return 0;
}
```

arrays

an array holds many variables of the same type; each is called an array `element`

```
int d[10]; // ten integers, stored as elements of an array called "d"
```

instead of giving each variable a name, it is given a numeric *index* starting at 0

```
d[0] = 42; // first integer element in the array
d[9] = 99; // last integer element in the array
```

array elements can be used just like any other variable

```
d[1] += d[2];
for (d[3] = 0;  d[3] < 10;  ++d[3]) {
    printf("%d\n", d[3]);
}
```

note: d has 10 elements with indexes 0 to 9 (*not* 1 to 10!)

counting occurrences of each digit

```c
#include <stdio.h>

int main()
{
    int c;
    int d[10];  // ten variables to store digit counts
    d[0] = d[1] = d[2] = d[3] = d[4] = d[5] = d[6] = d[7] = d[8] = d[9] = 0;

    while ((c = getchar()) != EOF) {
        if (c == '0') ++d[0];
        if (c == '1') ++d[1];
        if (c == '2') ++d[2];
        if (c == '3') ++d[3];
        if (c == '4') ++d[4];
        if (c == '5') ++d[5];
        if (c == '6') ++d[6];
        if (c == '7') ++d[7];
        if (c == '8') ++d[8];
        if (c == '9') ++d[9];
    }
```

continued...

```c
    printf("0 %d\n", d[0]);
    printf("1 %d\n", d[1]);
    printf("2 %d\n", d[2]);
    printf("3 %d\n", d[3]);
    printf("4 %d\n", d[4]);
    printf("5 %d\n", d[5]);
    printf("6 %d\n", d[6]);
    printf("7 %d\n", d[7]);
    printf("8 %d\n", d[8]);
    printf("9 %d\n", d[9]);

    return 0;
}
```

initialising arrays, performing arithmetic on characters

an array can be initialised with a sequence of values inside braces

```
int d[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }; // ten integers, set to zero
```

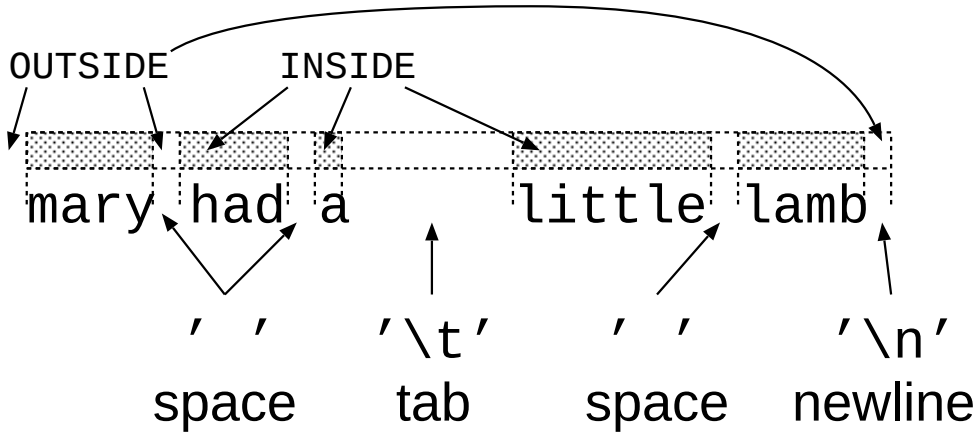
since characters are integers, you can perform arithmetic on them

```
if ('0' <= c && c <= '9')  // character in c is a digit
    ++d[c - '0'];          // convert char to array index in range [0..9]
```

any integer can be used as an array index

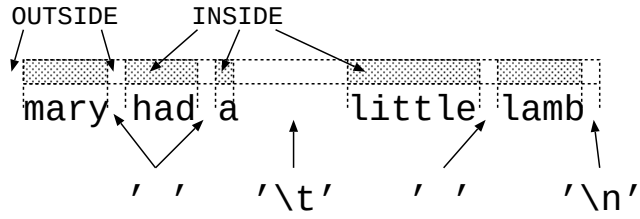
```
for (c = 0;  c < 10;  ++c)
    printf("%d was seen %d times\n", d[c]);
```

word counting



a new word begins at a transition from `OUTSIDE` to `INSIDE`

word counting



one (excellent) way to do this
is to use a *state machine*

```
let numWords be 0
let state be OUTSIDE                    // not in a word
while (read next character and it is not the end-of-file indicator)
    if the character is a space or a tab or a newline then
        let state be OUTSIDE            // we are not in a word
    else                                // we are in or starting a word
        if the state is OUTSIDE then    // we are starting a word
            add 1 to numWords           // count the word
            let state be INSIDE         // remember we are in a word
print numWords
```

next week: functions, line input, character arrays

1.7 Functions
1.8 Arguments-Call by Value
1.9 Character Arrays
1.10 External Variables and Scope

functions, `return`, call-by-value
character arrays
`getline` and character array `copy`
print longest line

exercises

please download exercises from

MS Team “Information Processing 2”, “General” channel

`IP2x02.pdf` plus supporting source/data files

exercises

use symbolic constants in temperature table

modify your copy program to replace multiple spaces with a single space

write a program that counts spaces, tabs, newlines

- modify it to detect the beginning of a word
- modify it to count the number of words

combine character count, word count, line count into a single program

- at the end, print the number of characters, words, lines read

write a program to print the numeric values of characters between `' '` and `'~'`

rewrite digit counting program to use

- an array initialiser to set all elements to `0` on one line
- integer arithmetic on digit characters to create an array index
- a `for` loop to print out the frequencies of the ten digits

print a histogram of digit frequencies (horizontally)

write a program to 'escape' tabs, backspaces, and backslashes

print numeric values of blank characters

print histogram of all character frequencies

12:40	15	last week: printf, while
12:55	10	for statement
13:05	10	exercise: temperature using for
13:15	5	symbolic constants
13:20	10	exercise: temperature constants
13:30	5	getchar, putchar
13:35	5	copying input to output
13:40	5	assignment is an expression
13:45	5	character counting
13:50	5	increment operator
13:55	5	line counting
14:00	5	if statement
14:05	5	line counting
14:10	10	*break*
14:20	5	logical operators
14:25	5	detecting spaces, tabs, newlines
14:30	5	word counting
14:35	5	detecting beginning of word
14:40	5	character arithmetic
14:45	5	frequency counts
14:50	5	arrays
14:55	5	initialising arrays
15:00	50	exercises
15:50	00	end