Computation I 5EIA0
C Exercises for Week 2
(v0.4, September 13, 2021)
Solutions

1 Introduction

The solutions are surrounded by the following standard code.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
int main (void) {
  // code is inserted here
}
```

2 Basic data types

1. Write a program that declares integer, float, and double variables, with initial values 1, 2, 3, respectively. Print them out like this:

```
integer 1
float   2.000000
double  3.000000e+00
```

Hint: use the `"%d"`, `"%f"`, `"%e"` `printf` format strings. (Appendix B1.2 of K&R has a full description.)

```
int i = 1;
float f = 2;   // automatically converted to 2.0
double d = 3;  // automatically converted to 3.0
printf("integer %d\n", i);
printf("float   %f\n", f);
printf("double  %e\n", d);
```

2. Write a program that declares integer and float variables. Ask for a value for each and print them out.

```
integer? 1
float? -2e3
integer +000001
float   -2000.00
float   -2.00e+03
```

Hint: use the `"%d"`, `"%f"`, `scanf` format strings. (Appendix B1.3 of K&R has a full description.) You will also need to *field width* and *precision* fields in the `printf` format string (see App. B1.2 of K&R).

```
int i;
float f;
printf("integer? ");
scanf("%d", &i);
printf("float? ");
scanf("%f", &f);
printf("integer %+06d\n", i);
printf("float   %06.2d\n", f);
printf("float   %+06.2e\n", f);
```

3. Write a program that declares a character variable. Read a single character from the terminal using `scanf`, *but skip white space*. White space characters are space, tab, and newline. Print the character in single quotes `'`. What happens when you have leading spaces, tabs, or newlines?

```
character?             <-- 3 leading spaces and a newline
  a                    <-- another 2 leading spaces
character 'a'
```

Hint: you can do this by using the `" %c"` format string. Notice the single space before the `%c`.

You will use `scanf(" %c", &c);` in almost every homework and exam. So it's worthwhile remembering it (and to understand the difference with the next program).

```
char c;
printf("character? ");
scanf(" %c", &c);
printf("character '%c'\n", c);
```

4. Write a program that declares a character variable. Read a single character from the terminal *without skipping white space* and print it in single quotes `'`. What happens when you enter newline/return as the character? Why?

```
character? a
character 'a'
```

Hint: use the `"%c"` `scanf` format string. You can also use `getchar()`.

```
char c;
printf("character? ");
scanf("%c", &c);
// or c = getchar();
printf("character '%c'\n", c);
```

When you enter newline the function `scanf` will read the newline character `'\n'` and print it, like so:

```
character?                    <-- entered a newline
character '
'
```

3 Integer arrays

1. Define a constant `N` with the value 6, using a `#define`. Write a program that declares an array of `N` integers all initialised with 0. Print the array with a loop.

```
0,0,0,0,0,0,
```

```
#define N 6
int a[N] = { 0, 0, 0, 0, 0, 0 };
// int a[] = { 0, 0, 0, 0, 0, 0 }; // the length 6 is computed automatically
// int a[N] = { 0 }; // the remaining values are initialised to 0 automatically
for (int i=0; i < N; i++) {
  printf("%d,", a[i]);
}
printf("\n");
```

2. Define a constant `N` with the value 6, using a `#define`. Write a program that declares an array of `N` integers, and ask the user for the values. Print the array with a loop.

```
value 0? 1
value 1? 10
value 2? -3
value 3? 6
value 4? 3
value 5? 4
1,10,-3,6,3,4,
```

```
#define N 6
int a[N]; // unitialised, contains random values
for (int i=0; i < N; i++) {
  printf("value %d? ", i);
  scanf("%d", &a[i]);
}
for (int i=0; i < N; i++) {
  printf("%d,", a[i]);
}
printf("\n");
```

3. Change the previous program to print the array without the trailing comma `,`.

```
value 0? 1
value 1? 10
value 2? -3
value 3? 6
value 4? 3
value 5? 4
1,10,-3,6,3,4
```

```
#define N 6
int a[N]; // unitialised, contains random values
for (int i=0; i < N; i++) {
  printf("value %d? ", i);
  scanf("%d", &a[i]);
}
for (int i=0; i < N; i++) {
  if (i != N-1)
    printf("%d,", a[i]);
  else
    printf("%d\n", a[i]);
}
// alternatively:
for (int i=0; i < N-1; i++) printf("%d,", a[i]);
printf("%d\n", a[N-1]);
// be careful not to accidentally use a[N], which is undefined (and wrong)
```

4. Using a constant `N` with the value 6, write a program that reads `N` integers into an array, and prints the smallest and smallest, and their sum.

```
value 0? 1
value 1? 10
value 2? -3
value 3? 6
value 4? 3
value 5? 4
1,10,-3,6,3,4,
min=-3
max=10
sum=21
```

```
#define N 6
int a[N]; // unitialised, contains random values
int min, max, sum;
for (int i=0; i < N; i++) {
  printf("value %d? ", i);
  scanf("%d", &a[i]);
}
min = a[0];
max = a[0];
sum = 0;
for (int i=0; i < N; i++) {
  printf("%d,", a[i]);
  if (a[i] < min) min = a[i];
  if (a[i] > max) max = a[i];
  sum = sum + a[i];
}
printf("\n");
printf("min=%d\n", min);
printf("max=%d\n", max);
printf("sum=%d\n", sum);
```

5. Using a constant `N` with the value 6, write a program that reads `N` integers into an array. Declare another array and store the numbers in reverse order in that array and print it. Change `N` to 7. Does your program still work?

```
value 0? 1
value 1? 10
value 2? -3
value 3? 6
value 4? 3
value 5? 4
4,3,6,-3,10,1,
```

```
#define N 6
int a[N];
int reverse[N];
for (int i=0; i < N; i++) {
  printf("value %d? ", i);
  scanf("%d", &a[i]);
}
for (int i=0; i < N; i++) {
  reverse[N-1-i] = a[i];
}
for (int i=0; i < N; i++) {
  printf("%d,", reverse[i]);
}
printf("\n");
```

6. Ask for the length of an array until it is between 1 and 100. Read all entries in the array from the terminal. Print all even numbers in the array in reverse order, but without using a second array.
   Hints: 1) Since you know that the array is at most 100 integers you can declare an array with 100 elements and use only those that you need. 2) Use a `do/while` loop to ask for the length. 3) When printing either use a loop from `length-1` to 0, or use `length-1-i` in the array index. 4) Remember the modulo operator. 5) To make it easier, you can print every number on a line by itself, as shown below.

```
length? 0
length? 6
value 0? 1
value 1? 10
value 2? -3
value 3? 6
value 4? 3
value 5? 4
4
6
10
```

```
#define N 100
int a[N];
int length = 0;
do {
  printf("length? ");
  scanf("%d", &length);
} while (length < 1 || length > 100);
for (int i=0; i < length; i++) {
  printf("value %d? ", i);
  scanf("%d", &a[i]);
}
for (int i=length-1; i >= 0; i--) {
  if (a[i] % 2 == 0) printf("%d\n", a[i]);
}
```

4 Character arrays

1. Define a constant `N` with the value 6, using a `#define`. Write a program that declares an array of `N` characters. Read the `N` characters (without skipping white space) from the terminal and print them enclosed in single quotes. (Hint: use `scanf("%c", &a[i]);` to read a single character.)

```
characters? 123456
'1', '2', '3', '4', '5', '6',
```

Can you explain this (correct) behaviour?

```
characters? 123
456
'1','2','3','
','4','5',
```

```
#define N 6
char a[N];
printf("characters? ");
for (int i=0; i < N; i++) {
  scanf("%c", &a[i]);
}
for (int i=0; i < N; i++) {
  printf("'%c',", a[i]);
}
printf("\n");
```

```
characters? 123
456
'1','2','3','
','4','5',
```

The carriage return character `\n` is read by the scanf function and saved in the array.

2. Write a program that reads `N=6` characters into an array. Print all characters that are digits (`'0'` to `'9'`).

```
characters? 12 4x5
1245
```

```
#define N 6
char a[N];
printf("characters? ");
for (int i=0; i < N; i++) {
  scanf("%c", &a[i]);
}
int digits = 0;
for (int i=0; i < N; i++) {
  if (a[i] >= '0' && a[i] <= '9') {
    printf("'%c',", a[i]);
    digits++;
  }
}
if (digits > 0) printf("\n");
```

3. Write a program that reads N=6 characters into an array. Compute and print the decimal number corresponding to the characters that are digits. (You can either assume that all characters are digits or skip them.)

```
characters? 123456
123456
```

Build up the program in steps. First, compute the number equivalent of the digit character. In other words, convert e.g. the character '4' to the number 4. Second, add the digit to the current number. Third, multiply the number by 10.

```
characters? 452967
convert character '4' to integer 4
number = number(0) + digit(4)
convert character '5' to integer 5
number = number(40) + digit(5)
convert character '2' to integer 2
number = number(450) + digit(2)
convert character '9' to integer 9
number = number(4520) + digit(9)
convert character '6' to integer 6
number = number(45290) + digit(6)
convert character '7' to integer 7
number = number(452960) + digit(7)
452967
```

```
#define N 6
char a[N];
int number = 0;
printf("characters? ");
for (int i=0; i < N; i++) {
  scanf("%c", &a[i]);
}
int debug = 0;
for (int i=0; i < N; i++) {
  if (a[i] >= '0' && a[i] <= '9') {
    number = number * 10;
    if (debug) printf("convert character '%c' to integer %d\n", a[i], a[i]-'0');
    if (debug) printf("number = number(%d) + digit(%d)\n", number, a[i]-'0');
    number += (a[i] - '0');
  }
}
printf("%d\n",number);
```

4. Write a program that reads N=6 characters into an array. Compute and print the *binary* number corresponding to the characters that are binary digits. (You can either assume that all characters are digits or skip them.) You only need to change three characters in your previous program! (Hint: changing base 10 to base 2 means changing the range of valid digits from 0-9 to 0-1, and multiplying by 2 instead of

10.)

```
characters? 111111
63
```

With debug output:

```
characters? 111011
convert character '1' to integer 1
number = number(0) + digit(1)
convert character '1' to integer 1
number = number(2) + digit(1)
convert character '1' to integer 1
number = number(6) + digit(1)
convert character '0' to integer 0
number = number(14) + digit(0)
convert character '1' to integer 1
number = number(28) + digit(1)
convert character '1' to integer 1
number = number(58) + digit(1)
59
```

```
#define N 6
char a[N];
int number = 0;
printf("characters? ");
for (int i=0; i < N; i++) {
  scanf("%c", &a[i]);
}
for (int i=0; i < N; i++) {
  if (a[i] >= '0' && a[i] <= '1')
    number = number * 2 + (a[i] - '0');
}
printf("%d\n", number);
```

5 Functions

When you are asked to write a function such as `int plus(int a, int b)` this means that the function is called `plus`, it has two integer inputs, called `a` and `b`, respectively, and returns an integer value.

1. Write a function `int plus2(int a, int b)` that returns the sum of its inputs `a` and `b`. In your `main` function ask for two integers, call `plus2` with those integers, and print the result.

```
int plus2 (int a, int b) ...
int main(void) {
  int x, y;
  printf("integers? ");
  scanf("%d %d", &x, &y);
  ...
}
```

```
integers? 10 21
the sum is 31
```

```
int plus2 (int a, int b) {
  return a + b;
}
int main(void) {
  int x, y;
  printf("integers? ");
  scanf("%d %d", &x, &y);
  printf("the sum is %d\n", plus2(x,y));
}
```

2. What happens if you rename the variables `x` and `y` to `a` and `b`, respectively?
   Nothing, everything still works. The reason is that the integers `a` and `b` in the `main` function are defined only within the scope of the `main` function (i.e. between its curly braces). Similarly the integers `a` and `b` in the `plus` function are defined only within the scope of the `plus` function (i.e. between its curly braces). The integers `a` and `b` in the `plus` function are separate from those in the `main` function. They are created only when the `plus` function is called and are destroyed when the `plus` function finishes.

3. Write a function `int plus2(int a, int b)` that returns the sum of its inputs `a` and `b`. In your `main` function ask for `N=6` integers. Compute and print the sum of the integers using `plus2`.

```
integers? -1 -3 5 9 333 0
the sum is 343
```

```
#define N 6
int plus2 (int a, int b) {
  return a + b;
}
int main(void) {
  int a[N], sum = 0;
  printf("integers? ");
  for (int i=0; i < N; i++) {
    scanf("%d", &a[i]);
    sum = plus2(sum, a[i]);
  }
  printf("the sum is %d\n", sum);
}
```

4. Write functions `int plus2(int a, int b)` and `int plus3(int a, int b, int c)` that return the sum of their inputs. Use `plus2` to compute the sum in `plus3`. In your `main` function ask for `N=6` integers. Compute and print the sum of the integers using three calls to `plus3`.

```
integers? -1 -3 5 9 333 1
the sum is 344
```

```
#define N 6
int plus2 (int a, int b) {
  return a + b;
}
int plus3 (int a, int b, int c) {
  int ab = plus2(a, b);
  int abc = plus2(ab, c);
  return abc;
}
int main(void) {
  int a[N];
  printf("integers? ");
  for (int i=0; i < N; i++) {
    scanf("%d", &a[i]);
  }
  printf("the sum is %d\n", plus3(0, plus3(a[0], a[1], a[2]), plus3(a[3], a[4], a[5])));
  // of course, it would have been smarter to do:
  // printf("the sum is %d\n", plus2(plus3(a[0], a[1], a[2]), plus3(a[3], a[4], a[5])));
}
```

5. Write a functions `int plus6(int a[N])` that returns the sum of the array input. In your `main` function ask for `N=6` integers. Compute and print the sum of the integers using a call to `plus6`.

```
integers? 1 2 3 4 5 6
the sum is 21
```

```
#define N 6
int plus6 (int a[N]) {
  int sum = 0;
  for (int i=0; i < N; i++) sum += a[i];
  return sum;
}
int main(void) {
  int a[N];
  printf("integers? ");
  for (int i=0; i < N; i++) {
    scanf("%d", &a[i]);
  }
  printf("the sum is %d\n", plus6(a));
}
```

6. Write a function `int plusn(int a[], int n)` that returns the sum of the array input. Since the array length is not given in the array bound `int a[]`, the integer `n` indicates the length of the array. In your `main` function ask for `N`=6 integers. Compute and print the sum of the integers using a call to `plusn`. (Hint: the sum of n down to 0 is equal to n plus the sum of n-1 down to 0.)

```
integers? 1 2 3 4 5 6
the sum is 21
```

```
#define N 6
int plusn (int a[], int n) {
  int sum = 0;
  for (int i=0; i < n; i++) sum += a[i];
  return sum;
}
int main(void) {
  int a[N];
  printf("integers? ");
  for (int i=0; i < N; i++) {
    scanf("%d", &a[i]);
  }
  printf("the sum is %d\n", plusn(a, N));
}
```

7. Extend your program to print the sum of elements like this, using a loop with a call to `plusn`.

```
integers? 1 2 3 4 5 6
the sum of the first 1 elements is 1
the sum of the first 2 elements is 3
the sum of the first 3 elements is 6
the sum of the first 4 elements is 10
the sum of the first 5 elements is 15
the sum of the first 6 elements is 21
```

```
#define N 6
int plusn (int a[], int n) {
  int sum = 0;
  for (int i=0; i < n; i++) sum += a[i];
  return sum;
}
int main(void) {
  int a[N];
  printf("integers? ");
  for (int i=0; i < N; i++) {
    scanf("%d", &a[i]);
  }
  for (int i=1; i <= N; i++) {
    printf("the sum of the first %d elements is %d\n", i, plusn(a, i));
  }
}
```

8. Write a function `int vectormult(int a[N], int b[N])` that returns $\sum_i a_i * b_i$, i.e. the sum of the product of the elements. In your `main` function ask for `N`=6 integers for each of `a` and `b`. Call and print the result of `vectormult`.

```
a? 1 2 3 4 5 6
b? 1 2 3 4 5 6
the product is 91
```

```
#define N 6
int vectormult (int a[N], int b[N]) {
  int sum = 0;
  for (int i=0; i < N; i++) sum += (a[i] * b[i]);
  return sum;
}
int main(void) {
  int a[N], b[N];
  printf("a? ");
  for (int i=0; i < N; i++) scanf("%d", &a[i]);
  printf("b? ");
  for (int i=0; i < N; i++) scanf("%d", &b[i]);
  printf("the product is %d\n", vectormult(a, b));
}
```

9. Modify your `vectormult` to use your previously defined `plus2` function.

```
a? 1 2 3 4 5 6
b? 1 2 3 4 5 6
the product is 91
```

```
#define N 6
int plus2 (int a, int b) {
  return a + b;
}
int vectormult (int a[N], int b[N]) {
  int sum = 0;
  for (int i=0; i < N; i++) sum = plus2(sum, (a[i] * b[i]));
  return sum;
}
int main(void) {
  int a[N], b[N];
  printf("a? ");
  for (int i=0; i < N; i++) scanf("%d", &a[i]);
  printf("b? ");
  for (int i=0; i < N; i++) scanf("%d", &b[i]);
  printf("the product is %d\n", vectormult(a, b));
}
```

10. Write a function `float powr(float b, int e)` that returns the floating point number `b` raised to the power `e`. Ask for a float and an int in the main function, and print the result of calling `pow`.

```
base? 3
exponent? 0
the result is 1.000000
base? 3
exponent? 5
the result is 243.000000
base? 1.001
exponent? 65
the result is 1.067127
```

```
float powr (float b, int e) {
  float r = 1;
  for (int i=0; i < e; i++) r *= b;
  return r;
}
int main(void) {
  float b;
  int e;
  printf("base? ");
  scanf("%f", &b);
  printf("exponent? ");
  scanf("%d", &e);
  printf("the result is %f\n", powr(b, e));
}
```

11. Now make `powr` *recursive*, i.e. call itself to compute the product! (Hint: $b^e = b * b^{e-1}$.)

```
base? 3
exponent? 0
the result is 1.000000
base? 3
exponent? 5
the result is 243.000000
base? 1.001
exponent? 65
the result is 1.067127
```

```
float powr (float b, int e) {
  if (e < 1) return 1.0;
  float powr_e_minus_one = powr(b, e-1);
  return b * powr_e_minus_one;
}
int main(void) {
  float b;
  int e;
  printf("base? ");
  scanf("%f", &b);
  printf("exponent? ");
  scanf("%d", &e);
  printf("the result is %f\n", powr(b, e));
}
```

12. Write a function `int sumn(int n)` that returns the sum of all integers from 0 up to and including `n`. `sumn` must be *recursive*, i.e. call itself to compute the sum! Ask for `n` in your main program and print the result of calling sumn.

```
sum up to? 1111
the sum is 617716
```

```
int sumn (int n) {
  if (n < 1) return 0;
  int sum_of_0_to_n_minus_one = sumn(n-1);
  return sum_of_0_to_n_minus_one + n;
}
int main(void) {
  int n;
  printf("sum up to? ");
  scanf("%d", &n);
  printf("the sum is %d\n", sumn(n));
}
```

13. Read a string using `scanf` and the `"%s"` format string. Implement your own `strlen` function that returns the length of a string.

```
string? xx
length is 2
```

```
int mystrlen(char s[]) {
  int len = 0;
  while (s[len] != '\0') len++;
  return len;
}
int main (void) {
  char string[100];
  printf("string? ");
  scanf("%s", &string[0]);
  printf("length is %d\n", mystrlen(string));
```

14. Read a string using `scanf` and the `"%s"` format string. Implement your own `strpy` function that copies the string (second argument) to another string (first argument).

```
string? xx
string1="xx"
string2="xx"
```

```
void mystrcpy(char s[], char t[]) {
  int len = 0;
  while (from[len] != '\0') {
    to[len] = from[len];
    len++;
  }
}
int main (void) {
  char string1[100] = "you should not see this printed 1";
  char string2[100] = "you should not see this printed 2";
  printf("string? ");
  scanf("%s", &string[0]);
  mystrcpy (to, from);
  printf("string1=\"%s\"", string1);
  printf("string2=\"%s\"", string2);
```

Note that the function assumes that the `from` string is null-terminated and that the `to` string has sufficient space.

6 Operators

1. Write a program that prints 2^i for i from 0 to 32. Use the bitwise shift operator (see K&R Section 2.9).

```
0000000001
0000000002
0000000004
...
1073741824
2147483648
```

```
for (int i=0; i < 32; i++) printf("%010u\n", 1 << i);
```

2. Write a program that asks for an unsigned integer and prints out which bits are '1' in its binary representation. Use bitwise operators, not math functions such as `pow`.

```
number? 60
bit 2 is 1
bit 3 is 1
bit 4 is 1
bit 5 is 1
```

```
int u;
printf("number? ");
scanf("%u", &u);
for (int i=0; i < 32; i++) {
  if (u & (1 << i)) printf("bit %d is 1\n",i);
}
```

3. Write a function `void print_binary(unsigned int)` that prints out the 32-bit binary representation of the unsigned integer (without the newline). The main program asks for an integer and calls the function. Use bitwise operators, not math functions such as `pow`.

```
number? 60
00000000000000000000000000111100
```

```
void print_binary(unsigned int u) {
  // start with the most significant bit (31)
  for (int i=31; i >= 0; i--) {
    if (u & (1 << i)) printf("1"); else printf("0");
    // or: printf("%c", '0' + ((u & (1 << i)) != 0));
  }
}
int main(void) {
  int u;
  printf("number? ");
  scanf("%u", &u);
  print_binary(u);
  printf("\n");
}
```

4. Write a function that asks for a 32-bit unsigned binary number. Call it in the main function and print the number and its one's complement. Use the previously written `print_binary` function. (Hint: with `scanf` read a string of characters into an array, which you can then convert to an unsigned integer. If you use `" %c"` as the `scanf` format string then it skips white space, which makes it easier to enter the input.)

```
32-bit binary number? 0000 0000 0000 0000 0000 0000 1001 1110
00000000000000000000000010011110
11111111111111111111111101100001
```

```
void print_binary(unsigned int u) {
  for (int i=31; i >= 0; i--) if (u & (1 << i)) printf("1"); else printf("0");
}
unsigned int read_binary(void) {
  char s[32];
  unsigned int u;
  printf("32-bit binary number? ");
  // start with the most significant bit (31)
  for (int i=31; i >= 0; i--) {
    // use " %c" to skip spaces
    scanf(" %c", &s[i]);
    if (s[i] >= '0' && s[i] <= '1') {
      u = 2*u + s[i] - '0';
    } else {
      printf("error: '%c' is not a binary digit\n", s[i]);
    }
  }
  return u;
}
int main (void) {
  unsigned int u = read_binary();
  print_binary(u);
  printf("\n");
  print_binary(~u);
  printf("\n");
}
```

5. Write a program that asks for two 32-bit unsigned binary numbers and prints their bitwise AND, OR, and exclusive OR. (Output is wrapped for legibility.)

```
32-bit binary number? 0000 0000 0000 0000 0000 0000 0000 0011
32-bit binary number? 0000 0000 0000 0000 0000 0000 1001 1110
00000000000000000000000000000011 & 00000000000000000000000010011110 =
00000000000000000000000000000010
00000000000000000000000000000011 | 00000000000000000000000010011110 =
00000000000000000000000010011111
00000000000000000000000000000011 ^ 00000000000000000000000010011110 =
00000000000000000000000010011101
```

```
void print_binary(unsigned int u) { /* see above */ }
unsigned int read_binary(void) { /* see above */ }
int main (void) {
  unsigned int u1, u2;
  u1 = read_binary();
  u2 = read_binary();
  print_binary(u1);
  printf(" & ");
  print_binary(u2);
  printf(" = ");
  print_binary(u1 & u2);
  printf("\n");
  print_binary(u1);
  printf(" | ");
  print_binary(u2);
  printf(" = ");
  print_binary(u1 | u2);
  printf("\n");
  print_binary(u1);
  printf(" ^ ");
  print_binary(u2);
  printf(" = ");
  print_binary(u1 ^ u2);
  printf("\n");
}
```

7 More advanced character input

1. Write a program that repeatedly reads a character from the terminal. Print each character on a line by itself in single quotes. Use ^C (control-C) to stop the program. Run the program with inputs that include the newline/return, spaces, tabs, etc. Can you explain what's going on?

```
characters? hell o
'h'
'e'
'l'
'l'
' '
'o'
'
'
stop!
's'
't'
'o'
'p'
'!'
'
'
^C
```

```
char c;
printf("characters? ");
while (1) {
  scanf("%c", &c);  // or c = getchar();
  printf("'%c'\n", c);
}
```

Explanation:

```
characters? hell o      <-- the input you type
'h'                     <-- each character is printed on a line
'e'
'l'
'l'
' '                     <-- including the space ' ' character
'o'
'                       <-- and the newline'\n' character
'
stop!                   <-- a new line of input from you
's'
't'
'o'
'p'
'!'
'
'
^C                      <-- type control-C to stop the program
```

2. Write a program that reads exactly *one line* from the input and prints it out, one character at a time.

```
one line? hello there
hello there
```

Hint: use a `do/while` loop that contains a `scanf` and a `printf` statement.

```
char c;
printf("one line? ");
do {
  scanf("%c", &c);  // or c = getchar();
  printf("%c", c);
} while (c != '\n');
```

3. Now add double quotes when you print the line.

```
one line? hello there
"hello there"
```

Your first version may have looked like this; what went wrong?

```
one line? "hello there
hello there
"
```

(Hint: first, only print the trailing double quote correctly. You'll need an `if` statement to avoid printing the newline `'\n'`. After you get this working then, print the leading double quote but only after the `scanf`, for which you will need to know that you are printing the first character on the line. Use a variable `first` for this.)

Likely first incorrect version:

```
char c;
printf("one line? ");
printf("\"");
do {
  scanf("%c", &c);  // or c = getchar();
  printf("%c", c);
} while (c != '\n');
printf("\"\n");
```

Correct version:

```
char c;
int first = 1;
printf("one line? ");
do {
  scanf("%c", &c);  // or c = getchar();
  if (first) printf("\"");
  first = 0;
  if (c != '\n') printf("%c", c);
} while (c != '\n');
printf("\"\n");
```

4. Now keep reading lines (until you type ^C).

```
one line? hello there
"hello there"
one line?    going going       <-- note that are 3 leading and 1 trailing space
"   going going "              <-- as can be seen here
one line? gone!
"gone!"
one line?
^C
```

```
char c;
do {
  printf("one line? ");
  int first = 1;
  do {
    scanf("%c", &c);  // or c = getchar();
    if (first) printf("\"");
    first = 0;
    if (c != '\n') printf("%c", c);
  } while (c != '\n');
  printf("\"\n");
} while (1);
```

5. Write a program that reads a line of text and prints whether each character comes before or after the letter 'k' in the alphabet. You only need to do this for the letters 'a' to 'z'. (Hint: use character comparisons and arithmetic as described on p23 of K&R.)

```
one line? hello
'h' comes 3 letters earlier in the alphabet than 'k'
'e' comes 6 letters earlier in the alphabet than 'k'
'l' comes 1 letters later in the alphabet than 'k'
'l' comes 1 letters later in the alphabet than 'k'
'o' comes 4 letters later in the alphabet than 'k'
```

```
char c;
printf("one line? ");
do {
  scanf("%c", &c);  // or c = getchar();
  if (c >= 'a' && c <= 'z') {
    if (c < 'k') printf("'%c' comes %d letters earlier in the alphabet than 'k'\n", c, 'k' - c)
    if (c > 'k') printf("'%c' comes %d letters later in the alphabet than 'k'\n", c, c - 'k');
  }
} while (c != '\n');
```

6. Write a program (called `rot1`) that reads a line of text and rotates the letters 'a' to 'z' in the alphabet. Thus 'a' becomes 'b', 'b' becomes 'c', etc. You have to ensure that 'z' becomes 'a'. All other characters are unchanged. (Hint: use character comparisons and arithmetic as described on p23 of K&R.)

```
one line? abc xyz
bcd yza
```

```
char c;
printf("one line? ");
do {
  scanf("%c", &c);  // or c = getchar();
  if (c >= 'a' && c < 'z') printf("%c", c+1);
  else if (c == 'z') printf("a");
  else printf("%c", c);
} while (c != '\n');
```

7. In case you used an `if` statement to treat `'z'` differently from the other letters, remove it in a new version of your program. (Hint: use the modulo operator to wrap around.)

```
one line? abc xyz
bcd yza
```

```
char c;
printf("one line? ");
do {
  scanf("%c", &c);  // or c = getchar();
  if (c >= 'a' && c <= 'z')
    printf("%c", 'a' + ((c - 'a') + 1) % 26);
  else
    printf("%c", c);
} while (c != '\n');
```

- **19/1 v0.1** First version.
- **13/9 v0.4** Fixes.