Information Processing 2 — Week 09 exercises

Complete the first five exercises in-class on your own laptop. Ask an instructor to check your work during the class as soon as you finish answering each question. A **late penalty of 50%** will be applied to answers checked after class.

Complete the remaining exercises in-class if possible, or before the start of the next exercise class at latest. Exercises checked **more than one week** after the class will have a **50% late penalty** applied.

1 Pointers to strings

Write a function `int strlength(char *s)` that counts the number of characters in the string `s` (excluding the final 'nul' character). Use pointer dereferencing to test each character and use pointer increment to traverse the string. Count the number of characters using an `int` variable initialised to `0`. Do not use array indexing in your answer.

Test your function with the following program (download '`09-01-strlength.c`' from Teams):

```
int main()
{
    printf("%d\n", strlength("hello, world")); // 12
    char s[] = "hello, world";
    printf("%d\n", strlength(s));              // 12
    char *p = s;
    printf("%d\n", strlength(p));              // 12
    return 0;
}
```

Verify that it prints '12' three times.

▷▷ Ask an instructor to check and record your work.

2 Pointer arithmetic

Write another version of `int strlength(char *s)` that does not use an integer variable. Instead make a copy of the value of `s` then search for the terminating 'nul' character. Subtract the original value of `s` from the location of the 'nul' character to obtain the length of the string. Verify your function with the same program as the previous question.

▷▷ Ask an instructor to check and record your work.

3 Copying strings with pointers

Write a function `char *strcopy(char *s, char *t)` that copies the string pointed to by `t` into the memory pointed to by `s`. Assume there is enough space for the string at the destination address. Return the location of the copy (i.e., the original value of `s`).

Test your function with the following program (download '`09-03-strcopy.c`' from Teams):

```
int main()
{
    char s[100] = "overwrite", *t = "me!";
    printf("%s\n", strcopy(s, t));
    return 0;
}
```

Verify that the output is "`me!`".

▷▷ Ask an instructor to check and record your work.

4 Combining string operations

Write a function `char *strappend(char *s, char *t)` that appends the string `t` to the string `s`, then returns the original value of `s`.

Hints: a simple version of this function uses a `while` loop to find the end of `s` and then a second while loop to append `t` to the end of `s`. Another solution would be to call `strlength(s)` to find the length of `s` and then add that value to `s` to find the address of the end of `s` where strcopy should copy `t`. You can choose either solution.

Test your function with the following program (download '`09-04-strappend.c`' from Teams):

```
int main()
{
    char s[100] = "append ", *t = "me!";
    printf("%s\n", strappend(s, t));
    return 0;
}
```

Verify that the output is "`append me!`".

▷▷ Ask an instructor to check and record your work.

5 Longest of two strings

Write a macro `MAX(A, B)` whose value is the larger of `A` or `B`.

Use `MAX()` to write a function `int strmax(char *s, char *t)` that returns the length of the longer of the strings `s` and `t`. Use `strlength()` to measure the lengths of the strigs. Do not call `strlength()` more than once for each string.

Test your function with the following program (download '`09-05-strmax.c`' from Teams):

```
int main()
{
    printf("%d\n", strmax("", ""));
    printf("%d\n", strmax("", "abc"));
    printf("%d\n", strmax("abc", ""));
    printf("%d\n", strmax("abc", "abc"));
    printf("%d\n", strmax("abcdef", "abc"));
    printf("%d\n", strmax("abc", "abcdefghi"));
    return 0;
}
```

Verify that the output is: "0 3 3 3 6 9"

▷▷ Ask an instructor to check and record your work.

6 Swapping two strings

Write a function `void swapchars(char *c, char *d)` that swaps the character at location `c` with the one at `d`. Use that function to write `void swapstrings(char *s, char *t)` that swaps the entire contents of the strings pointed to by `s` and `t`.

Hints: one way to solve this is to use `strmax()` to find the longer of `s` and `t`, and then use `swapchars()` to swap that number of characters between strings `s` and `t`. You will swap some irrelevant characters at the end of the shorter string, but you can ignore that detail for now.

Test your functions on the following program (download '`09-06-swapstrings.c`' from Teams):

```
int main()
{
    char s[100] = "the first string";
    char t[100] = "another string, a bit longer";
    printf("%s\n%s\n", s, t);
    swapstrings(s, t);
    printf("%s\n%s\n", s, t);
    return 0;
}
```

Verify that the output is:

```
the first string
another string, a bit longer
another string, a bit longer
the first string
```

▷▷ Ask an instructor to check and record your work.

7 Reversing a string using pointers

You already wrote a function `reverse(char s[])` that reverses the characters in the string `s`. Write a version `char *reverse(char *s)` that reverses the characters in `s` using pointers, then returns the original value of `s`. Do not use array indexing in your answer.

Hint: the algorithm can be the same as before. Find the last character in the string. While the location of the last character is greater than the location of the first character, swap the last and first characters and then decrement the last location and increment the first location. Your loop will finish when the first and last pointers meet (or cross by one character). You can assume the existence of `strlength()` and `swapchars()` to help.

Test your function with the following program (download '`09-07-reverse.c`' from Teams):

```c
int main()
{
    char a[100] = "";
    char b[100] = "b";
    char c[100] = "bc";
    char d[100] = "bcd";
    char s[100] = "peter piper picked a peck";
    printf("%s\n", reverse(a));
    printf("%s\n", reverse(b));
    printf("%s\n", reverse(c));
    printf("%s\n", reverse(d));
    printf("%s\n", reverse(s));
    return 0;
}
```

Verify that the output is (with an initial blank line):

```
b
cb
dcb
kcep a dekcip repip retep
```

▷▷ Ask an instructor to check and record your work.

8 Reversing a range of array elements

Write a function `char *revchars(char *s, char *t)` that reverses a string starting with the character pointed to by `s` up to, but not including, the character pointed to by `t`. Return the original value of `s`. You can again use `swapchars()` to help.

Test your function using the following program (download '`09-08-revchars.c`' from Teams):

```c
int main()
{
    char a[100] = "";
    char b[100] = "b";
    char c[100] = "bc";
    char d[100] = "bcd";
    char s[100] = "peter piper picked a peck";
    printf("%s\n", revchars(a, a+strlength(a)));
    printf("%s\n", revchars(b, b+strlength(b)));
    printf("%s\n", revchars(c, c+strlength(c)));
    printf("%s\n", revchars(d, d+strlength(d)));
    printf("%s\n", revchars(s, s+strlength(s)));
    return 0;
}
```

Verify that the output is the same as the last question.

▷▷ Ask an instructor to check and record your work.

9 Reversing individual words in a line

Write a function `char *revwords(char *s)` that reverses each individual word in the string `s`, then returns the original value of `s`. Hints: one way to solve this is to write a loop that identifies the location of the first character of each word in `s`, and the location of the first character after the word (perhaps using a function `wordlen()` that counts the number of letters in memory at a given address, and adding the result to the location of the first character). When you have those locations, `revchars()` will reverse just that word for you.

Test your function with the following program (download '`09-09-revwords.c`' from Teams):

```
int main()
{
    char s[100] = "retep repip dekcip a kcep fo delkcip reppep";
    printf("%s\n", revwords(s));
    return 0;
}
```

Verify that the output is: `peter piper picked a peck of pickled pepper`

10 Secret codes

A simple (but not very secure) secret code is easily made by 'rotating' each letter in a message 13 places in the alphabet. An 'a' (the first letter in the alphabet) becomes a 'n' (the 13th letter, 'b' becomes 'o', and so on. Of course, the upper case letters need to be processed too. One advantage of this encoding is that applying it twice to a string yields the original string (it is its own inverse).

Write a function `char *rot13(char *s)` that rotates each letter in `s` by 13 places in the alphabet and then returns the original value of `s`. Rotate both upper and lower case letters. When rotating, be careful to wrap around from 'z' back 'a'. Hints: A `switch` statement is perfect for detecting letters. One way (out of many) to rotate a letter is to add 13 to it and then adjust by 26 if the result went past the end of the alphabet.

Test your function with the following program (download '`09-10-rot13.c`' from Teams):

```
int main()
{
    char s[] =
        "Fvzcyvpvgl naq ryrtnapr ner hacbchyne orpnhfr gurl erdhver uneq jbex naq "
        "qvfpvcyvar gb npuvrir naq rqhpngvba gb or nccerpvngrq. -- Rqftre Qvwxfgen";
    printf("%s\n", rot13(s));
    return 0;
}
```

Verify that the output is a complete, correct sentence in English.

11 Challenge: Reversing the order of lines in a file

Write a program to reverse the order of lines in a file. Each individual line should remain unchanged but they should be printed out starting with the last line and finishing with the first. For example, if the input is:

```
One
Two
Buckle my shoe
```

then the output would be:

```
Buckle my shoe
Two
One
```

You can assume some reasonable limits such as no more than 1000 characters per line and no more than 1000 lines in total. Hints:

- You will need `getchars()` to read lines. (Rewrite it to use pointers, if you like.)
- A fixed size `char line[]` buffer can be used to read input lines.
- A function `char *strduplicate(char *s)` will be very useful. It duplicates the string `s` by copying it into new memory. To obtain new memory you can `#include <stdlib.h>` at the start of your program and then call `malloc(size)` which will return a pointer to `size` bytes of empty memory into which you can `strcopy()` the current input `line`.
- Don't forget to allocate one more byte than the length of the string you are copying, for the terminating nul which *must* be there.
- Copy each input line, as above, and store it in an array of pointers to characters, which you can declare as `char *lines[LINESMAX]` to store up to `LINESMAX` input lines. 1000 is a reasonable upper limit.
- When there are no more lines to read, print out the copied lines stored in `lines[]` starting with the highest index and working back towards 0, to reproduce the contents of the input with the lines in reverse order.

Test your program with any convenient text file.

▷▷ Ask an instructor to check and record your work.