Computation I 5EIA0
C Exercises for Week 4
(v0.3, August 2, 2022)
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
}
```

Note that the addresses that your code prints are most likely different from the ones shown in the example outputs. That is because compilers on different computers may place variables at different addresses in memory. This is why you should never use absolute addresses.

2 Pointers and Addresses

1. Write a program that declares integer, float, and double variables, with initial values 1, 2, 3, respectively. Print out their values and addresses like shown below. The addresses are most likely different.

```
i=1 at address 0x7ffcfa1f4b18
f=2.000000 at address 0x7ffcfa1f4b1c
d=3.000000e+00 at address 0x7ffcfa1f4b20
```

   Hint: use the "%p" printf format string to print out a pointer.

```
  int i = 1;
  float f = 2;
  double d = 3;
  printf("i=%d at address %p\n",i,&i);
  printf("f=%f at address %p\n",f,&f);
  printf("d=%e at address %p\n",d,&d);
```

2. Write a program that declares an integer array of length 3 initialised to {1,2,3} and a character array of length 5 initialised to "hell". Print out all values like shown below. Explain why the addresses increment in the way they do.

```
i[0]=1 at address 0x7ffefe705a84
i[1]=2 at address 0x7ffefe705a88
i[2]=3 at address 0x7ffefe705a8c
c[0]='h' at address 0x7ffefe705a93
c[1]='e' at address 0x7ffefe705a94
c[2]='l' at address 0x7ffefe705a95
c[3]='l' at address 0x7ffefe705a96
c[4]='' at address 0x7ffefe705a97
```

```
  int i[3] = {1,2,3};
  char c[5] = "hell";
  for (int j=0; j < 3; j++)
    printf("i[%d]=%d at address %p\n",j,i[j],&i[j]);
  for (int j=0; j < 5; j++)
    printf("c[%d]='%c' at address %p\n",j,c[j],&c[j]);
```

The addresses of the elements of the integer array increment by 4 since an integer takes 4 bytes (32 bits) in memory. The addresses of the elements of the character array increment by 1 since a character takes 1 byte (8 bits) in memory. Also, recall that the first element of the array must be aligned in memory. (In fact, this holds for any variable.) Thus the integer addresses are multiples of four.

3. Write a program that declares an integer variable i and two variables p1 and p2 of type pointer to integer. Assign 10 to i. Assign the address of i to both p1 and p2. Print out i, p1, and p2, but also the address of p1 and the address of p2. Why are the values of p1 and p2 the same, but their addresses different?

```
i=10
&i=0x7ffc274dd924
p1=0x7ffc274dd924
p2=0x7ffc274dd924
&p1=0x7ffc274dd928
&p2=0x7ffc274dd930
```

```
  int i = 10;
  int *p1 = &i;
  int *p2 = &i;
  printf("i=%d\n",i);
  printf("&i=%p\n",&i);
  printf("p1=%p\n",p1);
  printf("p2=%p\n",p2);
  printf("&p1=%p\n",&p1);
  printf("&p2=%p\n",&p2);
```

p1 and p2 are variables that happen to contain the same value (namely the address of i). But since they are different variables they must be stored in different locations in memory, which is why &p1 and &p2 (i.e. their addresses) are different.

4. Modify the previous program such that you change the value of i to 20, but without a direct assignment to i. Print out new value of i. Hint: use p1 or p2!

```
i=10
&i=0x7ffc274dd924
p1=0x7ffc274dd924
p2=0x7ffc274dd924
&p1=0x7ffc274dd928
&p2=0x7ffc274dd930
i=20
```

```
  int i = 10;
  int *p1 = &i;
  int *p2 = &i;
  printf("i=%d\n",i);
  printf("&i=%p\n",&i);
  printf("p1=%p\n",p1);
  printf("p2=%p\n",p2);
  printf("&p1=%p\n",&p1);
  printf("&p2=%p\n",&p2);
  *p1 = 20;
  // or: *p2 = 20;
  printf("i=%d\n",i);
```

5. Write a program that declares an integer array of length 7 initialised to `{7,6,5,4,3,2,1}`. Print out the values of the array using `a[i]`. Then print them out again but without using array indexing. Instead, use a variable p of type pointer to integer that points to the first element of the array. Then add 0, 1,

etc. to p to print out the elements of the array. You should see that adding i to an integer pointer to get the ith element in the array increments it by the size of an integer (i*4 bytes).

```
a[0]=7
a[1]=6
a[2]=5
a[3]=4
a[4]=3
a[5]=2
a[6]=1
p=0x7ffd96fcdec0+0=0x7ffd96fcdec0 a[0]=7
p=0x7ffd96fcdec0+1=0x7ffd96fcdec4 a[1]=6
p=0x7ffd96fcdec0+2=0x7ffd96fcdec8 a[2]=5
p=0x7ffd96fcdec0+3=0x7ffd96fcdecc a[3]=4
p=0x7ffd96fcdec0+4=0x7ffd96fcded0 a[4]=3
p=0x7ffd96fcdec0+5=0x7ffd96fcded4 a[5]=2
p=0x7ffd96fcdec0+6=0x7ffd96fcded8 a[6]=1
```

```
  int a[7] = {7,6,5,4,3,2,1};
  int *p = &a[0];
  for (int i=0; i < 7; i++)
    printf("a[%d]=%d\n",i,a[i]);
  for (int i=0; i < 7; i++)
    printf("p=%p+%d=%p a[%d]=%d\n", p, i, p+i, i, *(p+i));
```

3 Pointers and Function Arguments

1. Write a function swap that swaps the values of two floating point numbers. Use this as your main program:

```
int main(void)
{
  double a = 10, b = 20;
  printf("a=%f b=%f\n", a, b);
  swap (....);
  printf("a=%f b=%f\n", a, b);
}
```

```
a=10.000000 b=20.000000
a=20.000000 b=10.000000
```

```
void swap (double *x, double *y)
{
  double s = *x;
  *x = *y;
  *y = s;
}
  ... swap (&a, &b); ...
```

2. Write a function swap that swaps the values of two floating point numbers. Use this as your main program. Notice that the arguments in the function call to swap have changed.

```
int main(void)
{
  double a = 10, b = 20;
  // declare pa & pb
  printf("a=%f b=%f\n", a, b);
  swap (pa, pb);
  printf("a=%f b=%f\n", a, b);
}
```

```
a=10.000000 b=20.000000
a=20.000000 b=10.000000
```

```
void swap (double *x, double *y)
{
  double s = *x;
  *x = *y;
  *y = s;
}
int main(void)
{
  double a = 10, b = 20;
  double *pa = &a, *pb = &b;
  printf("a=%f b=%f\n", a, b);
  swap (pa, pb);
  printf("a=%f b=%f\n", a, b);
}
```

3. Write a function `next_element` that takes an array `a[10]` and an integer i as input and return a pointer to an integer. The return value points to the (i+1)th element in the array. If the `a[i]` is the last element of the array, then the function should return the first element.

   (a) This is the type of the function: `int *next_element (int a[10], int i)`
   (b) What is the address of the ith element of the array?
   (c) How can you compute the address of the (i+1)the element of the array?
   (d) Use a modulo operator to wrap around.

   Use this as your main program:

```
int *next_element (int a[10], int i) { ... your code ... }
int main(void)
{
  int a[10] = {10,9,8,7,6,5,4,3,2,1};
  for (int i=0; i < 10; i++) {
    int *next = next_element(a,i);
    printf("a[%d]=%d is stored at %p; the next element a[%d]=%d is stored at %p\n",
           i, a[i], &a[i], i, *next, next);
  }
}
```

```
a[0]=10 is stored at 0x7fffb2620e70; the next element a[0]=9 is stored at 0x7fffb2620e74
a[1]=9 is stored at 0x7fffb2620e74; the next element a[1]=8 is stored at 0x7fffb2620e78
a[2]=8 is stored at 0x7fffb2620e78; the next element a[2]=7 is stored at 0x7fffb2620e7c
a[3]=7 is stored at 0x7fffb2620e7c; the next element a[3]=6 is stored at 0x7fffb2620e80
a[4]=6 is stored at 0x7fffb2620e80; the next element a[4]=5 is stored at 0x7fffb2620e84
a[5]=5 is stored at 0x7fffb2620e84; the next element a[5]=4 is stored at 0x7fffb2620e88
a[6]=4 is stored at 0x7fffb2620e88; the next element a[6]=3 is stored at 0x7fffb2620e8c
a[7]=3 is stored at 0x7fffb2620e8c; the next element a[7]=2 is stored at 0x7fffb2620e90
a[8]=2 is stored at 0x7fffb2620e90; the next element a[8]=1 is stored at 0x7fffb2620e94
a[9]=1 is stored at 0x7fffb2620e94; the next element a[9]=10 is stored at 0x7fffb2620e70
```

```
int *next_element(int a[10], int i)
{
  // (i+1) % 10 gives the next element, wrapping around the end
  // a[(i+1)%10] is that element in the array
  // &a[(i+1)%10] is the address of that element
  return &a[(i+1)%10];
  // alternatively, just work with the address:
  return a+(i+1)%10;
```

4. Write a function that computes the length of a string (not including the null character). Do not use array indexing `s[i]` but only pointer variables. Use this as your main program:

```
int len(char *s) { ... your code ... }
int main(void)
{
  char s[100] = { '\0' };
  printf("String? ");
  scanf("%s",s);
  // same as scanf("%s", &s[0]);
  printf("len(\"%s\")=%d\n", s, len(s));
}
```

```
computation@computation-virtual-machine:~$ ./a.out
String? string
strlen("string")=6
computation@computation-virtual-machine:~$ ./a.out
String? very long string
strlen("very")=4
computation@computation-virtual-machine:~$ ./a.out
String? very-long-string
strlen("very-long-string")=16
```

```
int len(char *s)
{
  int l = 0;
  while (*s != '\0') { l++; s++; }
  // or: while (*s) { l++; s++; }
  // or: while (*s++) l++;
  return l;
}
```

5. Write a function that reverses a string. Do not use array indexing `s[i]` but only pointer variables address arithmetic (address +/- integer). Use your own string length function. Use a temporary array for the reversed string. Do not use recursion. Use this as your main program:

```
int len(char *s) { ... your code ... }
void rev(char *s) { ... your code ... }
int main(void)
{
  char s[100] = { '\0' };
  printf("String? ");
  scanf("%s",s);
  printf("rev(\"%s\")=", s);
  rev(s);
  printf("%s\n", s);
}
```

(Hint: use the previously written function len(char *s) to calculate the length of the string)

```
String? hello
rev("hello")=olleh
```

```
void rev(char *s)
{
  int l = len(s);
  if (l == 0) return;
  char copy[l];
  for (int i = 0; i < l; i++) *(copy+l-1-i) = *(s+i);
  // now copy the reversed string back into s
  for (int i = 0; i < l; i++) *(s+i) = *(copy+i);
}
```

6. Write a function that reverses a string. Do not use array indexing `s[i]` but only pointer variables and increment (++) and decrement (--) operators. Use your own string length function. Use a temporary array for the reversed string. Do not use recursion. Use this as your main program:

```
int len(char *s) { ... your code ... }
void rev(char *s) { ... your code ... }
int main(void)
{
  char s[100] = { '\0' };
  printf("String? ");
  scanf("%s",s);
  printf("rev(\"%s\")=", s);
  rev(s);
  printf("%s\n", s);
}
```

```
String? hello
rev("hello")=olleh
```

```
void rev(char *s)
{
  int l = len(s);
  if (l == 0) return;
  char copy[l];
  char *sp = s;
  char *cp = copy+l-1;
  for (int i = 0; i < l; i++) {
    *cp-- = *sp++;
  }
  // now copy the reversed string back into s
  sp = s;
  cp = copy;
  for (int i = 0; i < l; i++) {
    *sp = *cp;
    sp++;
    cp++;
    // or: *sp++ = *cp++;
  }
}
```

This is an alternative that uses the terminating null character instead of the string length:

```
void rev(char *s)
{
  int l = len(s);
  if (l == 0) return;
  char copy[l];
  char *sp = s;        // start at beginning and increment
  char *cp = copy+l-1; // start at end and decrement
  while (*sp != '\0') { *cp = *sp; cp--; sp++; }
  // note that we have reversed UP TO the null character
  // now copy the reversed string back into s
  // note that we did not assign a null character to terminate the copy
  // that's fine because we use the length l in the loop below
  sp = s;
  cp = copy;
  while (l > 0) { *sp = *cp; sp++; cp++; l--; }
}
```

7. Write a function that reverses a string. Do not use array indexing `s[i]` but only pointer variables. Do not use a temporary array for the reversed string. (Hint: a single character to swap is sufficient.) Do not use recursion. Use this as your main program:

```
int len(char *s) { ... your code ... }
void rev(char *s) { ... your code ... }
int main(void)
{
  char s[100] = { '\0' };
  printf("String? ");
  scanf("%s",s);
  printf("rev(\"%s\")=", s);
  rev(s);
  printf("%s\n", s);
}
```

```
String? hello
rev("hello")=olleh
```

```
void rev(char *s){
  int l = len(s);
  for (int i = 0; i < l/2; i++) {
    char swap = *(s+i);
    *(s+i) = *(s+l-1-i);
    *(s+l-1-i) = swap;
  }
}
```

8. Write a function that reverses a string. Do not use array indexing `s[i]` but only pointer variables. Now use recursion. Use this as your main program:

```
int len(char *s) { ... your code ... }
void rev(char *s, int l) { ... your code ... }
int main(void)
{
  char s[100] = { '\0' };
  printf("String? ");
  scanf("%s",s);
  printf("rev(\"%s\")=", s);
  rev(s, len(s));
  printf("%s\n", s);
}
```

```
String? hello
rev("hello")=olleh
```

```
void rev(char *s, int l){
  if (l < 2) return;
  char swap = *s;
  *s = *(s+l-1);
  *(s+l-1) = swap;
  rev(s+1, l-2);
}
```

4 Pointers into Arrays

1. Declare an array `letters` of 53 characters. Initialise it with a loop with the characters a-z and then A-Z. Insert a null character at the last position. Fill in the missing code in the following main program:

```
int main(void)
{
  // declare letters, letter
  // initialise letters
  printf("letter? ");
  scanf(" %c", &letter);
  // your code
  printf("the start address of the array is %p\n", ...);
  printf("the array index of '%c' in the array is %d\n", letter, pos);
  printf("the address of '%c' in the array is %p\n", ...);
}
```

```
computation@computation-virtual-machine:~$ ./a.out
letter? a
the start address of the array is 0x7ffe8b726540
the array index of 'a' in the array is 0
the address of 'a' in the array is 0x7ffe8b726540
computation@computation-virtual-machine:~$ ./a.out
letter? A
the start address of the array is 0x7ffdb5974980
the array index of 'A' in the array is 26
the address of 'A' in the array is 0x7ffdb597499a
computation@computation-virtual-machine:~$ ./a.out
letter? Z
the start address of the array is 0x7ffc0e60a050
the array index of 'Z' in the array is 51
the address of 'Z' in the array is 0x7ffc0e60a083
computation@computation-virtual-machine:~$
```

```
  char letters[53] = { '\0' };
  char letter;
  for (int i=0; i < 26; i++) {
    letters[i] = 'a'+i;
    letters[i+26] = 'A'+i;
  }
  printf("letter? ");
  scanf(" %c", &letter);
  int pos;
  while (pos < 53 && letters[pos] != letter) pos++;
  printf("the start address of the array is %p\n", letters);
  printf("the array index of '%c' in the array is %d\n", letter, pos);
  printf("the address of '%c' in the array is %p\n", letter, letters + pos);
```

2. Extend the previous program with a declaration of four pointer to character variables: p1, p2, p3, p4. Set p1, p2, p3, p4 to the address of 'a', 'n', 'A', and 'N' in the array, respectively. Print the lengths of the strings p1, p2, p3, p4 as shown below, and explain.

```
letter? x
the start address of the array is 0x7ffc37841400
the array index of 'x' in the array is 23
the address of 'x' in the array is 0x7ffc37841417
p1=0x7ffc37841400 strlen(abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ)=52
p2=0x7ffc3784140d strlen(nopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ)=39
p3=0x7ffc3784141a strlen(ABCDEFGHIJKLMNOPQRSTUVWXYZ)=26
p4=0x7ffc37841427 strlen(NOPQRSTUVWXYZ)=13
```

```
  char letters[53] = { ’\0’ };
  char *p1 = letters + 0;
  char *p2 = letters + 13;
  char *p3 = letters + 26;
  char *p4 = letters + 39;
  char letter;
  for (int i=0; i < 26; i++) {
    letters[i] = ’a’+i;
    letters[i+26] = ’A’+i;
  }
  printf("letter? ");
  scanf(" %c", &letter);
  int pos;
  while (pos < 53 && letters[pos] != letter) pos++;
  printf("the start address of the array is %p\n", letters);
  printf("the array index of ’%c’ in the array is %d\n", letter, pos);
  printf("the address of ’%c’ in the array is %p\n", letter, letters + pos);
  printf("p1=%p strlen(%s)=%ld\n", p1, p1, strlen(p1));
  printf("p2=%p strlen(%s)=%ld\n", p2, p2, strlen(p2));
  printf("p3=%p strlen(%s)=%ld\n", p3, p3, strlen(p3));
  printf("p4=%p strlen(%s)=%ld\n", p4, p4, strlen(p4));
```

Note that we have to use `%ld` instead of `%d` in the last four printf since strlen returns a long int. Type "man strlen" in the terminal for additional information.

3. Absolute address calculations for p1 etc. are ugly. If you used them in the previous exercise (e.g. `char *p2 = letters + 13;`), now add a function `int find_pos(char *s, char c)` that returns the first position of c in s or the one plus the length of s otherwise. Call the new function to compute p1, p2, p3, and p4. You already have most of the code!

```
int find_pos (char *s, char c)
{
  int pos= 0;
  while (s[pos] != ’\0’ && s[pos] != c) pos++;
  return pos;
}
int main(void)
{
  char letters[53] = { ’\0’ };
  char letter;
  for (int i=0; i < 26; i++) {
    letters[i] = ’a’+i;
    letters[i+26] = ’A’+i;
  }
  // note that we moved these pi initialisations
  // because letters needs to be initialised first!
  char *p1 = letters + find_pos(letters, ’a’);
  char *p2 = letters + find_pos(letters, ’n’);
  char *p3 = letters + find_pos(letters, ’A’);
  char *p4 = letters + find_pos(letters, ’N’);
  printf("letter? ");
  scanf(" %c", &letter);
  printf("the start address of the array is %p\n", letters);
  printf("the array index of ’%c’ in the array is %d\n", letter, find_pos(letters,letter));
  printf("the address of ’%c’ in the array is %p\n", letter, letters + find_pos(letters,letter)
  printf("p1=%p strlen(%s)=%ld\n", p1, p1, strlen(p1));
  printf("p2=%p strlen(%s)=%ld\n", p2, p2, strlen(p2));
  printf("p3=%p strlen(%s)=%ld\n", p3, p3, strlen(p3));
  printf("p4=%p strlen(%s)=%ld\n", p4, p4, strlen(p4));
}
```

4. Modify the function to `char *find_pos(char *s, char c)`: it now returns the address of the first occurrence of c in s or the address of the null character otherwise. Make the necessary changes in your main program too.

```
letter? n
the start address of the array is 0x7ffc1fe4b7c0
the array index of ’n’ in the array is 13
the address of ’n’ in the array is 0x7ffc1fe4b7cd
p1=0x7ffc1fe4b7c0 strlen(abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ)=52
p2=0x7ffc1fe4b7cd strlen(nopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ)=39
p3=0x7ffc1fe4b7da strlen(ABCDEFGHIJKLMNOPQRSTUVWXYZ)=26
p4=0x7ffc1fe4b7e7 strlen(NOPQRSTUVWXYZ)=13
```

```
char *find_pos (char *s, char c)
{
  while (*s != ’\0’ && *s != c) s++;
  return s;
}
int main(void)
{
  char letters[53] = { ’\0’ };
  char letter;
  for (int i=0; i < 26; i++) {
    letters[i] = ’a’+i;
    letters[i+26] = ’A’+i;
  }
  // note that we moved these pi initialisations
  // because letters needs to be initialised first!
  char *p1 = find_pos(letters, ’a’);
  char *p2 = find_pos(letters, ’n’);
  char *p3 = find_pos(letters, ’A’);
  char *p4 = find_pos(letters, ’N’);
  printf("letter? ");
  scanf(" %c", &letter);
  printf("the start address of the array is %p\n", letters);
  printf("the array index of ’%c’ in the array is %ld\n", letter, find_pos(letters,letter) - le
  printf("the address of ’%c’ in the array is %p\n", letter, find_pos(letters,letter));
  printf("p1=%p strlen(%s)=%ld\n", p1, p1, strlen(p1));
  printf("p2=%p strlen(%s)=%ld\n", p2, p2, strlen(p2));
  printf("p3=%p strlen(%s)=%ld\n", p3, p3, strlen(p3));
  printf("p4=%p strlen(%s)=%ld\n", p4, p4, strlen(p4));
}
```

Note that we have to modify %d to `%ld` in the second printf because subtracting two pointers returns a long int.

5 Arrays of Pointers

1. Declare 3 integers i, j, k with initial values 11, 22, 33. Declare an array of integers a of length 3. Declare an array of pointers to integers pa of length 3. Set the elements of a to i, j, k, and the elements of pa to the addresses of i, j, k. Print the arrays like this:

```
a[0]=11
a[1]=22
a[2]=33
pa[0]=0x7ffee52498e8
pa[1]=0x7ffee52498e4
pa[2]=0x7ffee52498e0
*pa[0]=11
*pa[1]=22
*pa[2]=33
```

This is less complex than you may think: an array of integers is `int a[3]`. In other words, an array of 3 elements of type T is `T a[3];`. Since now we're asking for an array of 3 pointers to integers (`int *`) we should declare `int *a[3]`.

Similarly, you know to set an array element to a value: `a[0] = i`. It's exactly the same for an array of pointers: `pa[0] = something`. Of course, now the something is not the value of i but its address: `pa[0] = &i`.

```
int i = 11, j = 22, k = 33;
int a[3];
int *pa[3];
a[0] = i;
a[1] = j;
a[2] = k;
pa[0] = &i;
pa[1] = &j;
pa[2] = &k;
for (int x=0; x < 3; x++) printf("a[%d]=%d\n",x,a[x]);
for (int x=0; x < 3; x++) printf("pa[%d]=%p\n",x,pa[x]);
for (int x=0; x < 3; x++) printf("*pa[%d]=%d\n",x,*pa[x]);
```

2. Declare an array of doubles a of length 4 with value 0, 100, 200, 300. Declare an array of pointers to double pa of length 4. Set the ith elements of pa to the address of the ith element of a. Print the arrays like this:

```
a[0]=0.000000
a[1]=100.000000
a[2]=200.000000
a[3]=300.000000
pa[0]=0x7ffee18138f0
pa[1]=0x7ffee18138f8
pa[2]=0x7ffee1813900
pa[3]=0x7ffee1813908
*pa[0]=0.000000
*pa[1]=100.000000
*pa[2]=200.000000
*pa[3]=300.000000
```

```
double a[4];
double *pa[4];
for (int x=0; x < 4; x++) a[x] = 100*x;
for (int x=0; x < 4; x++) pa[x] = &a[x];
for (int x=0; x < 4; x++) printf("a[%d]=%f\n",x,a[x]);
for (int x=0; x < 4; x++) printf("pa[%d]=%p\n",x,pa[x]);
for (int x=0; x < 4; x++) printf("*pa[%d]=%f\n",x,*pa[x]);
```

3. Declare an array of characters a that is initialised with the string "another one bites the dust". Declare an array pa of 5 pointers to characters. Set the elements of the pointer array to the first letter of each word in the string (a, o, b, t, d). Print each element of pa as a pointer and as a string. Explain the output. Also print the characters a, o, b, t, d using the array pa (not the array a).

```
a="another one bites the dust"
pa[0]=0x7ffee94938f0
pa[1]=0x7ffee94938f8
pa[2]=0x7ffee94938fc
pa[3]=0x7ffee9493902
pa[4]=0x7ffee9493906
pa[0]="another one bites the dust"
pa[1]="one bites the dust"
pa[2]="bites the dust"
pa[3]="the dust"
pa[4]="dust"
*pa[0]='a'
*pa[1]='o'
*pa[2]='b'
*pa[3]='t'
*pa[4]='d'
```

```
char a[] = "another one bites the dust";
char *pa[5] = { &a[0], &a[8], &a[12], &a[18], &a[22] };
// or: char *pa[5] = { a+0, a+8, a+12, a+18, a+22 };
printf("a=\"%s\"\n",a);
for (int x=0; x < 5; x++) printf("pa[%d]=%p\n",x,pa[x]);
for (int x=0; x < 5; x++) printf("pa[%d]=\"%s\"\n",x,pa[x]);
for (int x=0; x < 5; x++) printf("*pa[%d]='%c'\n",x,*pa[x]);
```

- **22/7 v0.1** First version.
- **22/7 v0.2** Minor updates, added swap functions and pointer arrays.