Example 1: Function pointers can be used to register functions that will be called by the program at a later, determined time. The function `int atexit(void (*function)(void));` (from `stdlib.h`) may be used to register a function (or several, in successive calls) that will be called when the program terminates – either by returning from `main` or by calling `exit()`. The functions can't have parameters or results – so they can only access data through global variables. The following program uses such a function to print the time it has taken to execute.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void printtime(void)
{
  printf("Time used: %g s\n", (double)clock() / CLOCKS_PER_SEC);
}
int main(void)
{
  atexit(printtime);  // register function printtime to be called at the end
  for (int i = 1e8; i--;);   // dummy loop to consume time
  return 0;
}
```

In this simple case, the `clock()` function could have been just called at the end, directly; in general, the `atexit` mechanism allows doing some final action(s) (e.g., for cleanup or some program statistics) regardless of the way the program terminates (perhaps by calling `exit()` due to an error condition deep inside other functions).

Example 2 The same mechanism of *callback functions* (functions that are passed as parameters in order to be called at a later time) can be used *asynchronously*, as a reaction to certain events. For example, programs that run in windowing systems can react to a mouse click or a key pressed.

The GLUT Utility Toolkit built on top of OpenGL provides the function

```
void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));
```

which can be passed a pointer in order to register a function `func` that will be called everytime a key is pressed in the current window. When called, `func` will receive as parameters the character pressed and the coordinates of the mouse pointer within the window.

The following simple program uses this facility to draw a point colored depending on the key pressed (`r`, `g` or `b`) at the current mouse pointer location. Moving the mouse and pressing keys will generate a trail of colored points. To compile and run the program, the OpenGL and GLUT libraries must be installed. Compile the program with the options `-GL -lGLU -lglut` when using `gcc`.

```
#include <GL/glut.h>

void keypress(unsigned char key, int x, int y)
{
  switch(key) {       // set RGB color
  case 'r': glColor3f(1, 0, 0); break;
  case 'g': glColor3f(0, 1, 0); break;
  case 'b': glColor3f(0, 0, 1); break;
  case 'q': exit(0);
  }
  glBegin(GL_POINTS); // subsequent vertex commands will draw points
  glVertex2i(x, y);   // plot point in current color at mouse coordinates
  glEnd();            // end of drawing commands
  glFlush();          // force drawing of screen buffer
}
```

```
int main(int argc, char *argv[])
{
  glutInit(&argc, argv);
  glutCreateWindow(argv[0]);     // default: 300x300 pixels
  glClear(GL_COLOR_BUFFER_BIT); // clear color, default: black
  glFlush();
  gluOrtho2D(0, 300, 300, 0);    // set 2D viewing region for window
  glutKeyboardFunc(keypress);    // register function to be called on key press
  glutMainLoop();                // start event processing
  return 0;
}
```

Example 3: Function pointers may be used to parameterize certain algorithms. In addition to `qsort`, the standard C library also provides a function for binary search in an arbitrary sorted array.

We start by implementing such a function for a particular case, an array of real numbers (`double`). The function takes a number to search, the array an its size (assumed ≥ 1) and returns a pointer to the element if found in the array, or `NULL` otherwise.

```
double *binsrch(double x, double a[], size_t cnt)
{
  if (cnt == 1) return x == *a ? a : NULL;
  size_t half = cnt / 2;
  return x < a[half] ? binsrch(x, a, half) : binsrch(x, a + half, cnt - half);
}
```

If the array has size 1, the value is either at `*a` (i.e., `a[0]`) or does not exist. Else, the array is divided in half. If the value is less than the element at halfpoint, it is searched in the lower part of the array (excluding the halfpoint); else, it is searched in the upper part, starting at the halfpoint. Care must be taken to treat all boundary cases correctly. The code can also be easily written using a loop.

To make this code work with arbitrary arrays, we must make the following changes:

- the key to be searched for can no longer be given directly (since there is no type for arbitrary values in C), but through its address, a `void *` (pointer to unspecified type).
- the array and return value are likewise no longer given as a `double *` but as a `void *`
- elements can no longer be compared directly (with `==` or `<`) but with a comparison function passed as parameter. Like for `qsort`, the function takes two element addresses (as `void *`) and returns an `int` as the result of the comparison (`< 0`, `0`, or `> 0`).
- Since there is no pointer arithmetic on `void *`, the size of an array element has to be explicitly given, so the new boundaries can be computed (`a + half`)

With these changes, the code becomes:

```
void *bsearch(void *x, void *a, size_t cnt, size_t size,
                                            int (*cmp)(const void *, const void *))
{
  if (cnt == 1) return cmp(x, a) == 0 ? a : NULL;
  size_t half = cnt / 2;
  void *mid = (char *)a + half * size; // cast to char * for pointer arithmetic
  return cmp(x, mid) < 0 ?
        bsearch(x, a, half, size, cmp) : bsearch(x, mid, cnt - half, size, cmp);
}
```

This function is part of the C standard library (declared in `stdlib.h`). We will now discuss how to use it, highlighting the difference between an array of character arrays (matrix) and an array of pointers to strings.

For the first case, an example sorted array is

```
#define CNT    4
#define MAXL   6
char strmat[][MAXL] = {"four", "one", "three", "two" }; // array of arrays
```

The strings are stored in a contiguous memory region of 24 (4 * 6) characters. Pointer arithmetic: `strmat+0`, `strmat+1`, etc. produces `char (*)[6]` addresses which are `MAXL` (6) bytes apart, the strings are stored directly at these addresses. Since the comparison function receives directly the addresses where the strings are stored, we can use `strcmp` as-is, appropriately cast to fit as parameter for bsearch:

```
char *p = bsearch("foo", strmat, CNT, MAXL,        // search in matrix
                  (int (*)(const void *, const void *)strcmp);
```

A second case is when storing an array of pointers (similar to `argv[]` in `main`):

```
char *ptrarr[] = {"four", "one", "three", "two" };  // array of pointers
```

Here, the array itself does not contain strings. The elements `ptrarr[0]`, `ptrarr[1]`, etc. are addresses (`char *`); at these addresses in turn, we find the strings. Thus, the comparison function of `bsearch` will receive `char **` addresses (`ptrarr+0`, `ptrarr+1`, etc.) and we need to write a different comparison function that dereferences such a `char **` pointer to get a `char *` and pass it to `strcmp`:

```
int pstrcmp(const void *p1, const void *p2)
{
  return strcmp(*(char **)p1, *(char **)p2);
}
```

and the first parameter (the key searched) has also to be the *address* of a `char *` (string):

```
char **pp = bsearch(&"foo", ptrarr, CNT, sizeof(char *), pstrcmp);
```

Exercise 4: (suggested): Write a function that computes the root of a continuous function in a given interval using the bisection method (Lab 4, problem 2). The function whose root is computed is passed as parameter (together with the interval and the precision).

Exercise 5: (suggested): Write a program that draws any of the fractals discussed in class (box fractal, Koch curve, Sierpiński triangle, etc.), producing either an SVG or a PostScript file (given an option).

The two formats differ in the precise syntax to draw vector graphics, but the concepts are the same. In both cases we use four commands: *move* (the cursor) and *line* (from cursor to given point), where coordinates can be absolute or relative (to the cursor).

For SVG, the syntax is *command x y*, where *command* can be `M` or `L` (absolute), `m` or `l` (relative). For PostScript, the syntax is postfix: *x y command*, where the commands are `moveto`, `lineto`, `rmoveto` and `rlineto`.

For SVG, the minimal header is
`<?xml version="1.0"?>`
`<svg version="1.1" xmlns="http://www.w3.org/2000/svg">`
and the path to be drawn needs a drawing style besides the actual path element (`d=`):
`<path fill="none" stroke="black" d="` ... `"/>`
The closing tag `</svg>` ends the file.
A PostScript file starts with `%!PS`, a path starts with `newpath` and ends with `stroke`, and everything is displayed with `showpage`.

Define a structure that holds pointers to the four drawing functions, each taking two coordinates as parameters. Each of the drawing functions will print out the corresponding command, either in PostScript or SVG. The fractal function receives as parameter (a pointer to) the structure for the appropriate file type, depending on the option given. The standard beginning and ending text can be coded directly, or also through start/end functions, as you prefer.