function has been assigned to the pointer, we can use either one without any difference, i.e.,

```
entries = defineInput(data, MAX_NUM);
```

is identical to

```
pf = &defineInput;
entries = pf(data, MAX_NUM);
```

Introducing pointers to functions might seem very technical and of little importance. On the contrary, without this type of pointers it would be impossible to write functions executing algorithms on other functions which have not been defined beforehand. We discuss this interesting case in Section 7.6.

7.6 Functions of functions

There exist many situations in which it is useful, and sometimes necessary, to pass a function to another function as an input parameter. For example, suppose we want to translate the basic mathematical operators, such as the integral, the derivative, the summation, the product of a sequence operator, etc., into code. As mentioned in Section 7.5, we can pass a function to another one by pointers to a function. Once the syntactical rules have been understood, it is very easy to build this kind of application.

Let us consider, for example, the numeric computation of the derivative of a function. The derivative of a fuction $f(x)$ is defined as

$$f'(x) \equiv \frac{df(x)}{dx} \equiv \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}\,. \tag{7.2}$$

In equation (7.2) we need to compute the limit of the incremental ratio $\frac{f(x+\epsilon)-f(x)}{\epsilon}$. We can write a function computing this ratio numerically as a function of a variable parameter ϵ and define the numerical derivative as this incremental ratio. For sufficiently small ϵ, the ratio should coincide with $f'(x)$. One of the input arguments of this function computing the numerical derivative necessarily is a pointer to the function for which we want to compute the derivative.

The Listing 7.12 contains a possible way to create such a function, applied to the library function `sqrt(x)` computing the square root, $\sqrt{x}$.

In Listing 7.12 we defined a generic function `func(double x)` representing the function which is to be derived. In this specific example, the

function `func` simply returns `sqrt(x)`. However, it is clear that it could contain any possible complicated function.

```
#include <stdio.h>
#include <math.h>
double func(double);
double derivative(double (*)(double), double, double);

main() {
  double epsilon = 1.;
  printf("epsilon      derivata\n\n");
  while (epsilon > 1.e-18) {
    printf("%e %f\n", epsilon, derivative(func, 1., epsilon));
    epsilon /= 10.;
  }
}
double func(double x) {
  return sqrt(x);
}

double derivative(double (*f)(double), double x, double epsilon) {
  return ( f(x + epsilon) - f(x) ) / epsilon;
}
```

Listing 7.12 Computing the numerical derivative.

The function `derivative` computes the incremental ratio. Its input parameters are a pointer to the function `f`, for which we want to compute the derivative, the value of the variable with respect to which we want to perform the derivation, and the increment of the variable `epsilon`. In Listing 7.12 we assume `epsilon` is always larger than 0. Nevertheless, it would be appropriate to protect the `derivative` function against the occurrence of such a zero. In the `main` we start from $\epsilon = 1$ and divide it iteratively by 10. At each step we compute the value of the derivative and stop the iteration when $\epsilon \leq 10^{-18}$. We expect that for smaller and smaller values of ϵ the computed derivative is increasingly more precise. The result we obtained is listed in Table 7.6 and is somewhat surprising.

The given example is the computation of the derivative of $\sqrt{x}$ with respect to $x = 1$, which is equal to 0.5. It is clear from Table 7.6 that for values of $\epsilon \geq 10^{-5}$ the derivative is not yet correct because the value of ϵ is not yet small enough. When ϵ takes on a value comprised between 10^{-6} and 10^{-11}, the function always computes the correct value. For $10^{-15} \leq \epsilon < 10^{-11}$ the computed derivative moves farther and farther away form the correct value, until finally for $\epsilon < 10^{-15}$ the result is 0! The latter effect

ϵ	*derivative*
1.000 000e+00	0.414 214
1.000 000e−01	0.488 088
1.000 000e−02	0.498 756
1.000 000e−03	0.499 875
1.000 000e−04	0.499 988
1.000 000e−05	0.499 999
1.000 000e−06	0.500 000
1.000 000e−07	0.500 000
1.000 000e−08	0.500 000
1.000 000e−09	0.500 000
1.000 000e−10	0.500 000
1.000 000e−11	0.500 000
1.000 000e−12	0.500 044
1.000 000e−13	0.499 600
1.000 000e−14	0.488 498
1.000 000e−15	0.444 089
1.000 000e−16	0.000 000
1.000 000e−17	0.000 000
1.000 000e−18	0.000 000

is due to the fact that the difference between `f(x + epsilon)` and `f(x)` cancels out because of the limited precision of the numerical representation (remember that $10^{-16} \simeq 2^{-48}$): this is true when the derivative of the function is small, as is the case with $\sqrt{x}$. In particular, by printing the values of `x + epsilon`, `x`, `f(x + epsilon)` and `f(x)`, as they come along, we can verify that, even if the variable `epsilon` is different from 0, the computed function returns the same value in x as in $x + \epsilon$, such that its difference is zero. A similar effect may occur when the variable `epsilon` takes on values which are a lot less than the variable `x`. Using the same code of Listing 7.12, this effect can be verified by computing the derivative for $x = 1000$. In this case we dot not have enough bits for the mantissa of the variables `epsilon` and `x` to compute the incremental ratio in an adequate way. We conclude that a correct computation requires we do not exaggerate when reducing the increment ϵ, even if ϵ should be small, as this does not guarantee a greater precision.

Hands on 4 - Precision of the derivative

Verify the effect of the precision we just described by computing the derivative of the function $\sqrt{x}$ for increasing values of x, $x = 10, 100, 1000$, and printing the values of the variables `x + epsilon` and `x`, as well as `f(x + epsilon)` and `f(x)`, during the cycle. Compute the derivative of other more complicated functions numerically. Start for example with the function $\sin(x)$ for various values of x in the interval $[0, \frac{\pi}{2}]$. Next, try with a function of the type $(1 + x^2)^{-1}$ or similar. Study the behavior of the derivative when varying ϵ and check whether the obtained result is correct when ϵ lies in the same interval of values as the one of the example discussed in the text. Study whether the interval of values of ϵ for which a correct derivative is obtained depends on the fact that the function to be derived is "flat" or irregular.

Hands on 5 - Mathematical operators

Write the code for a summation function $\sum_k f_k(x)$, summing N terms $f_k(x)$, with N given by the user. Next, write a function computing the difference between the exact value of a function $f(x)$ and its expansion in Taylor series for a given value of x. For example, chose $f(x) = \sin x$ and a small value of x different from 0. Remember that $\sin x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$. Stop the series expansion at the third or fourth term.