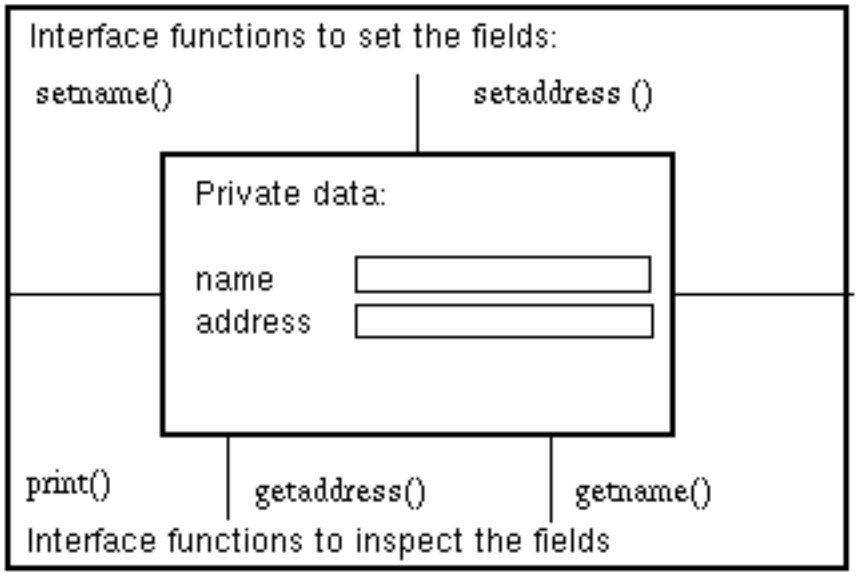

Figure 3.1: Private data and public interface functions of the class Person.

Another example of the concept of data hiding is the following. As an alternative to member functions which keep their data in memory (as do the above code examples), a runtime library could be developed with interface functions which store their data on file. The conversion of a program which stores `Person` structures in memory to one that stores the data on disk would not require any modification of the program using `Person` structures. After recompilation and linking the new object module to a new library, the program will use the new `Person` structure.

Though data hiding can be realized with `structs`, more often (almost always) classes are used instead. A `class` refers to the same concept as a struct, except that a `class` uses private access by default, whereas structs use public access by default. The definition of a `class Person` would therefore look exactly as shown above, except for the fact that instead of the keyword `struct`, `class` would be used, and the initial `private:` clause can be omitted. Our typographic suggestion for class names is to use a capital character as its first character, followed by the remainder of the name in lower case (e.g., `Person`).

3.6 Structs in C vs. structs in C++

Next we would like to illustrate the analogy between **C** and **C++** as far as `structs` are concerned. In **C** it is common to define several functions to process a `struct`, which then require a pointer to the `struct` as one of their arguments. A fragment of an imaginary **C** header file is given below:

```
// definition of a struct PERSON_
typedef struct
{
    char name[80];
    char address[80];
} PERSON_;
```