though we try to stick on the rules of structured programming. In particular, statements as `break`, `continue` and `switch` are subjects of discussion as to whether or not they can be used in a structured program. We do not enter this discussion, but rather avoid their use.

4.2 Taking a decision

The selection structure identified by the Böhm-Jacopini theorem allows to choose the portion of code that is to be executed depending on the truth value of a proposition. Let us call I_a and I_b two groups of mutually exclusive statements, and C the proposition which needs to be checked. The control structure can be symbolically represented by the flow charts of Figure 4.2 (a).

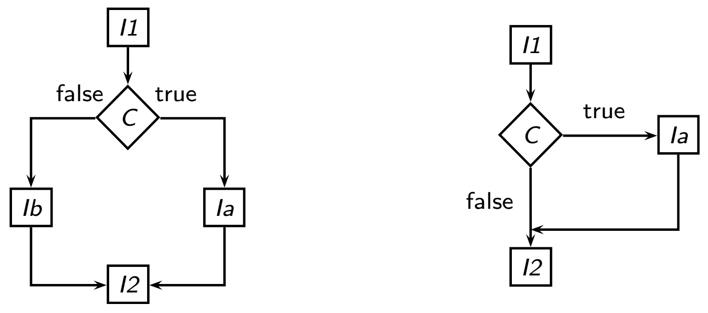

Fig. 4.2 (a) The general selection structure. (b) A simple selection structure, obtained from the fundamental one by substituting the I_b statement with the empty statement.

The program starts with the I_1 statement. Next, the truth value of the proposition C is evaluated: if it is true the group of statements I_a is executed, otherwise the block I_b is executed. At the end of the execution of one or the other set of instructions, the program continues by executing the statements I_2. The flow chart of Figure 4.2 (a) can be informally translated into: «If C is true, execute I_a, otherwise execute I_b».

The structure 4.2 (a) is the fundamental one, from which all others can easily be generated. For example, under some circumstances, the set of statements I_b is empty. In this case a more simple selection structure, not containing any alternatives, is generated, as the one shown in 4.2 (b). In this case the program starts by executing the statements I_1 and then evaluates the proposition C. If it results to be true, the commands in I_a are executed before the statements I_2 are executed. Otherwise, I_2 is executed immediately.

By combining several selection structures we obtain constructs allowing more complicated choices, as the one shown in Figure 4.3 (a). This is a chain of structures as the one shown in Figure 4.2 (a). By connecting the output of a first control structure with the input of a successive one, the block I_b is effectively substituted by a selection structure. Between the statements I_1 and I_2 a set of statements I_k are executed depending on whether one of the conditions C_j occurs. In case all conditions are false, the statements J are still executed (the set of these statements may be empty, of course). This structure corresponds to the one generated by the `switch` statement of the C language (Section 4.4). Another possibility, shown in Figure 4.3 (b), consists of a chain of non-mutual exclusive structures, generated by a sequence of the selection structures of Figure 4.2 (b), .

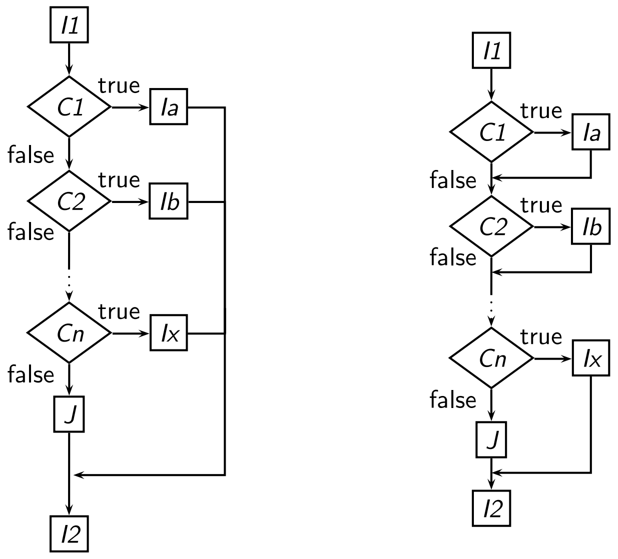

Fig. 4.3 To obtain multiple choice structures we can link several selection structures together. In the structure on the left the choices are mutually exclusive, while in the one on the right one choice does not exclude the others..

The difference between the two structures is only apparently small. Indeed, the program execution may vary drastically. In case of 4.3 (a), one true condition excludes all the others to occur. Once the statements corresponding to the true conditions are executed, the program immediately continues to execute the I_2 statements. Instead, in case of 4.3 (b), a true condition does not exclude others to occur. Thus, it is possible that several statement groups I_k are executed. Also, while in the former case only the conditions starting from C_1 up to the first true condition are evaluated, in the latter all conditions are evaluated. Thus, if the clauses are mutually exclusive, it is always convenient to choose a construct as the one shown in Figure 4.3 (a). Moreover, to improve performance, it is better to first insert the conditions which have a higher probability of occurring.

4.2.1 *if/else*

In C, the selection control structure can be achieved with the `if` construct, as shown in the Listing 4.1.

```
18  printf("Value in degrees Celsius= ");
19  scanf("%lf", &tc);
20  if (tc < -273.16) printf("Warning! T<0 K...\n");
```

Listing 4.1 A example of the `if` statement.

This Listing contains a part of the program of Listing 3.7. We inserted line 20, to check the value of the variable `tc`, which must not be smaller than −273.16 °C as it represents a temperature in degrees Celsius. The `if` statement on line 20 checks the value of `tc`: if it is smaller than −273.16, the program writes the message of the `printf` statement following it. The syntax of the `if` statement is the following:

```
if (expression) statement_1 [else statement_2]
```

When this statement is executed, the *expression* is evaluated and the corresponding result is interpreted as a logical value. If the expression is true, the program executes *statement_1*. Otherwise, if the `else` clause is present, *statement_2* is executed. The alternative statements can be compound, i.e., they may consist of several statements enclosed between curly brackets. Despite each statement must end with the character `;`, the latter is not required after the curly bracket as they mark off the end of a block.

A common error consists in inserting the `;` character after the logical condition[5]. In this case the compiler does not report any errors as the statement `if (`*expression*`);` *statement* is syntactically correct. It

[5]This error almost always occurs when a condition is followed by a single statement that is not enclosed between curly brackets (a good reason for using them...).

means that if the logical value of the `expression` is true, no statements are executed. Indeed, the semicolon signals the end of the `if` statement containing an empty statement. The next statement is interpreted as a normal statement following the `if`. Thus, it is executed even in case the logical value is false. Writing

```
if (tc < -273.16); printf("Warning! T<0 K...\n");
```

the program behaves the same way independently of the value of `tc` and always prints the warning message even if `tc` has an acceptable value. This error is usually avoided by using curly brackets. Therefore, we shall always write

```
if (tc < -273.16) {
   printf("Warning! T<0 K...\n");
}
```

Another common error consists in mixing up the assignment `=` and the comparison `==` operator. Indeed, quite often a condition like the following is included

```
if (a = 5) {
  ...statements...
}
```

In this case the error is due to the use of the assignment operator `=` instead of the comparison operator `==`. In C, every logical or mathematical expression has a value equal to the result of the expression. In this case, the `=` operator assigns the value 5 to the variable `a`. Therefore, the result of the expression included between parentheses is 5. In Section 1.6 we saw that the logical value *true* is represented in the memory by a bit sequence containing at least one 1. Therefore, in this case the expression is always equal to *true* and the condition is always satisfied. Moreover, the value of the variable `a` is always changed into 5 after this statement is executed. Note that from a syntactical point of view, the expression is correct. Indeed, the compiler would easily finish its job without warning about any possible disaster this expression might entail.

We now reformulate the Listing 3.7 with the selection control structure in order to avoid recompilation each time we want to change the type of conversion. Consider the Listing 4.2.

```
 1 #include <stdio.h>
 2
 3 main() {
 4   double tIn, tOut, offset, conv;
 5   int option = 0;
 6
 7   offset = 32.;
 8   printf("Press\n'1' to convert from F into C\n"
 9          "'2' to convert from C into F\n\nChoice:");
10   scanf("%d", &option);
11   if (option == 1) {
12     conv = 5./ 9.;
13     printf("Value in degrees Fahrenheit= ");
14     scanf("%lf",&tIn);
15     tOut = (tIn - offset) * conv;
16     printf("Value in degrees Celsius= %f\n",tOut);
17   } else {
18     conv = 9./ 5.;
19     printf("Value in degrees Celsius= ");
20     scanf("%lf", &tIn);
21     tOut = tIn * conv + offset;
22     printf("Value in degrees Fahrenheit= %f\n",tOut);
23   }
24 }
```

Listing 4.2 Temperature conversion with `if/else`.

First of all, note that we introduced an integer variable `option` allowing us to store a conventional value defining the type of conversion (from Fahrenheit into Celsius or vice versa). The value of the variable `option` is acquired on line 10, after an initial message looking like the following (try to understand why) appears on screen:

```
Press
'1' to convert from F into C
'2' to convert from C into F

Choice:
```

Note that the argument of `printf` on the preceding line is composed of a single string. Indeed, the C compiler interprets two succeeding character sets delimited by double quotes " as a single string (this technique is used to obtain a more readable program). On line 11, the value of `option` is compared to the constant 1. If the two values coincide, the lines starting from 12 up to line 16 are executed. Otherwise, the lines 18 up to 22 are executed. Even though it is not necessary, we changed the names `tc` and `tf` of the variables into `tIn` and `tOut` (*in* and *out* temperature). Indeed, in this context the meaning of the variables is different from the one in Listing 3.7. For coherence, it is appropriate to give names which reflect the correct meaning.

4.2.2 *The selection operator*

When the selection control is used to determine the value of an expression which depends on whether a condition occurs or not, it is sometimes preferable to use a single, ternary C operator: the operator `?`. For example, a structure determining the maximum between two values `a` and `b`, could be the following:

```
(a > b) ? (max = a) : (max = b);
```

First, the expression to the left of the question mark `(a > b)` is evaluated. If this expression is true, the expression to the left of the colon is evaluated. Otherwise, the one to the right. Note that we are not dealing with statements, but rather with expressions. Therefore, the semicolon and the parentheses are compulsory. Indeed, a completely analogue statement is

```
max = (a > b) ? a : b;
```

where the parentheses are not required as each expression simply consists of the variable's value. This statement can be read like this: «Is `a` larger than `b`? If yes, evaluate the value of `a`, otherwise that of `b`». The expression to the right of the `=` operator assumes the value of either `a` or `b` and the result is assigned to `max`.

4.3 Iterations

The third structure foreseen by the Böhm-Jacopini theorem is the iteration allowing the cyclic repetition of a block of statements. It can appear under various forms, which are all equivalent to each other. Indeed, it is possible to pass from one form to another by appropriately modifying the conditions. A first case is shown in Figure 4.4 (a). In this so-called *while-do* form, once the statement block I_1 has been executed, the condition C is checked. If the condition C is true, the block I_a is executed, at the end of which the program checks the validity of the condition C again. If it is still true, the statements I_a are executed again. At a certain point, they may alter