CH-230-A

Programming in C and C++

C/C++

Lecture 3

Dr. Kinga Lipskoch

Fall 2020

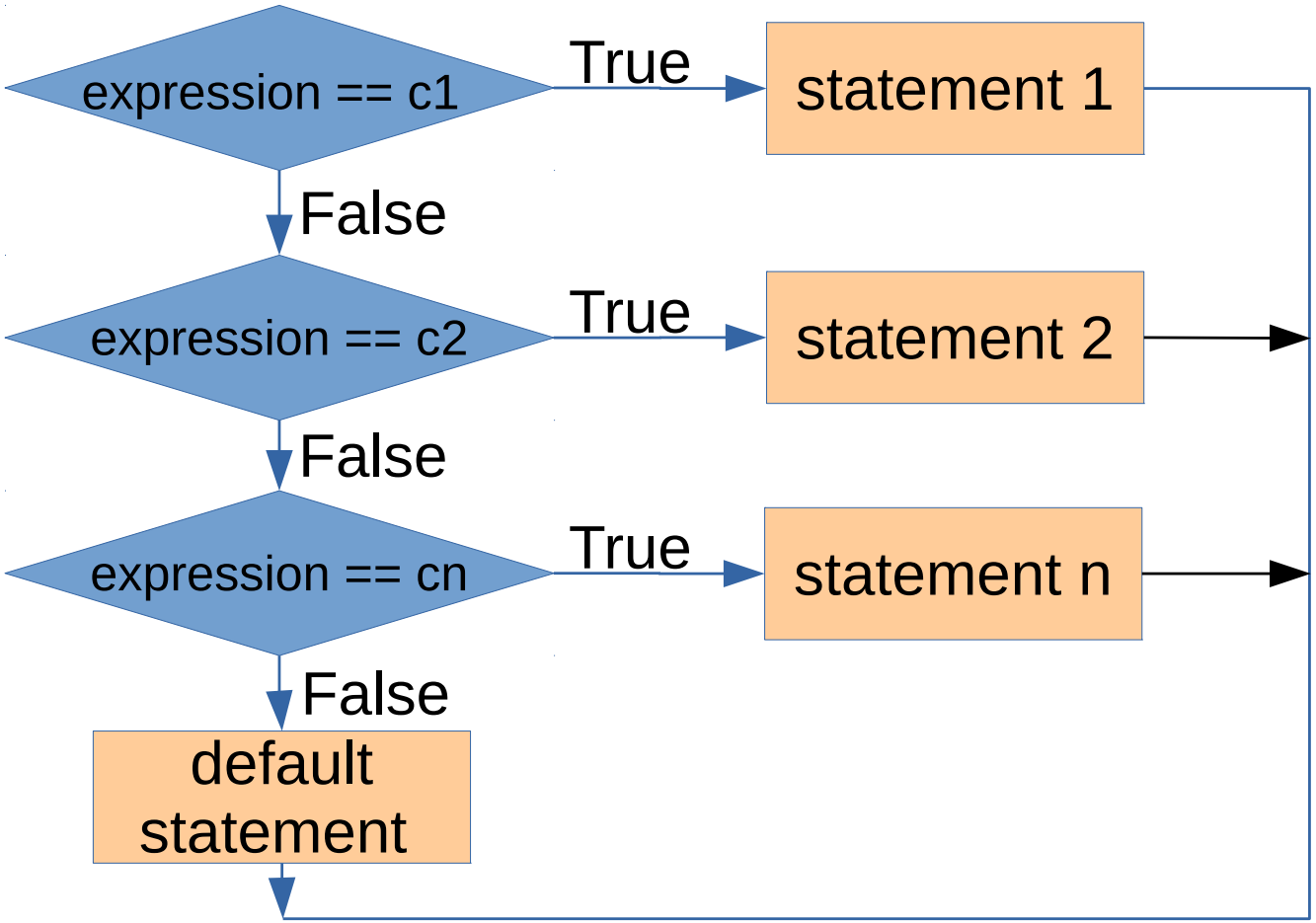
Multiple Choices: `switch`

- `switch` can be used when an expression should be compared with many values
- The same goal can be obtained with multiple `if`'s
- The expression must return an integer value

switch: The Syntax

```
1  switch (expression)  {
2    case c1:
3      statement1;
4      break;
5
6    case c2:
7      statement2;
8      break;
9
10   ...
11
12   default:
13     default_statement;
14 }
```

`switch`: Flow Chart



switch: Example

```c
#include <stdio.h>
int main() {
  int c;
  for (c = 0; c <= 3; c++) {
    printf("c: %d\n", c);

    switch (c) {
      case 1:
        printf("Here is 1\n");
        break;
      case 2:
        printf("Here is 2\n");
        /* Fall through */
      case 3:
      case 4:
        printf("Here is 3, 4\n");
        break;
      default:
        printf("Here is default\n");
    }
  }
  return 0;
}
```

Iterations

- In many cases it is necessary to repeat a set of operations many times
- Example: compute the average grade of the exam
  - Read all the grades, and sum them
  - Divide the sum by the number of grades
- C provides three constructs

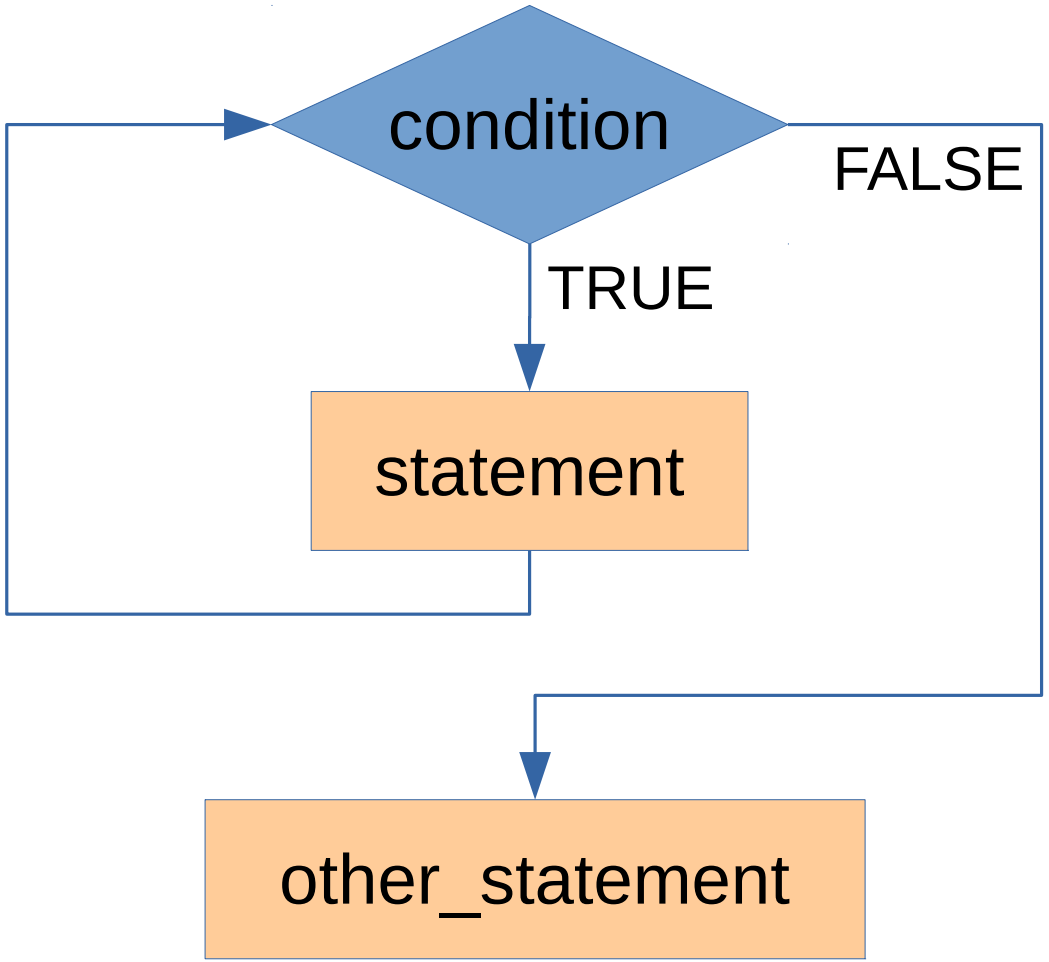
Iterations: `while`

- General syntax:

```
while (condition) {
  statement;
}
```

- Keep executing the statement as long as the condition is true

`while`: Flow Chart



Example:
Compute the Sum of the First n Natural Numbers

```c
#include <stdio.h>
int main() {
   int idx, n, sum = 0;
   printf("Enter a positive number ");
   scanf("%d", &n);
   idx = 1;
   while (idx <= n) {
      sum += idx;
      idx++;
   }
   printf("The sum is %d\n",sum);
   return 0;
}
```

Iterations: `for`

- General syntax:

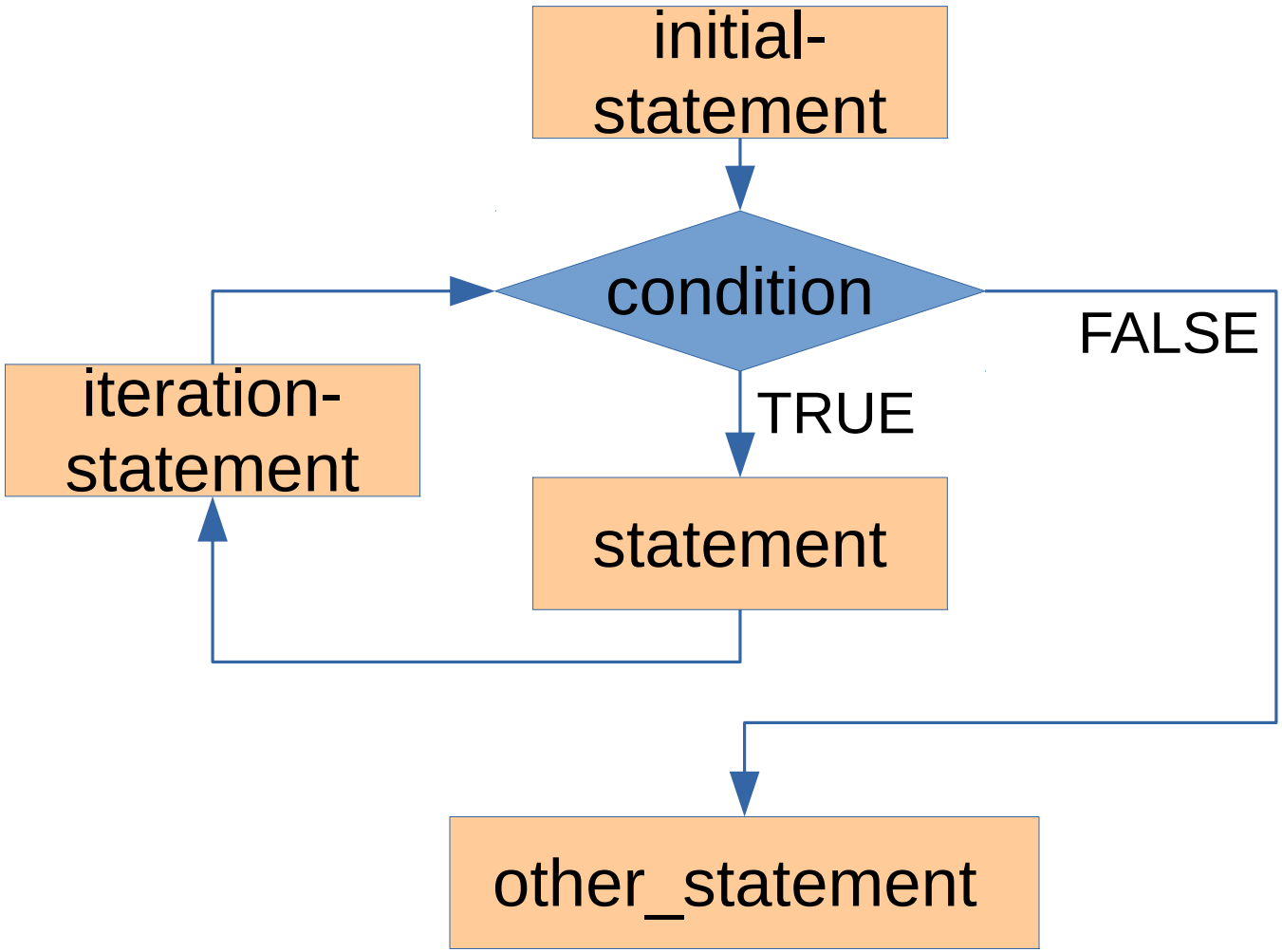
```
for (initial-statement; condition; iteration-
    statement)
  statement;
```

- Example:

```
for (n = 0; n <= 10; n++)
  printf("%d\n", n);
```

- The `for` and `while` loops can be made interchangeable

`for`: Flow Chart



for: Example Revised

```
1  #include <stdio.h>
2  int main() {
3    int idx, n, sum = 0;
4    printf("Type a positive number ");
5    scanf("%d", &n);
6    for (idx = 1; idx <= n; idx++) {
7      printf("Processing %d..\n", idx);
8      sum += idx;
9    }
10   printf("The sum is %d\n", sum);
11   return 0;
12 }
```

Boolean Operators and if

```
1  for (n = 0; n < 3; n++) {
2    for (i = 0; i < 10; i++) {
3      if (n < 1 && i == 0) {
4        printf("n is < 1, i is 0\n");
5      }
6      if (n == 2 || i == 5) {
7        printf("HERE n: %d i:%d\n", n, i);
8      }
9      else {
10       printf("n:%d, i:%d\n", n, i);
11     }
12   }
13 }
```

Easier or Harder to Read?

```
1 for (n = 0; n < 3; n++)
2   for (i = 0; i < 10; i++) {
3     if (n < 1 && i == 0) {
4       printf("n is < 1, i is 0\n"); }
5       if (n == 2 || i == 5) {
6        printf("HERE n: %d i:%d\n", n, i); }
7       else {
8         printf("n:%d, i:%d\n", n, i); }}}
```

Iterations: do ... while

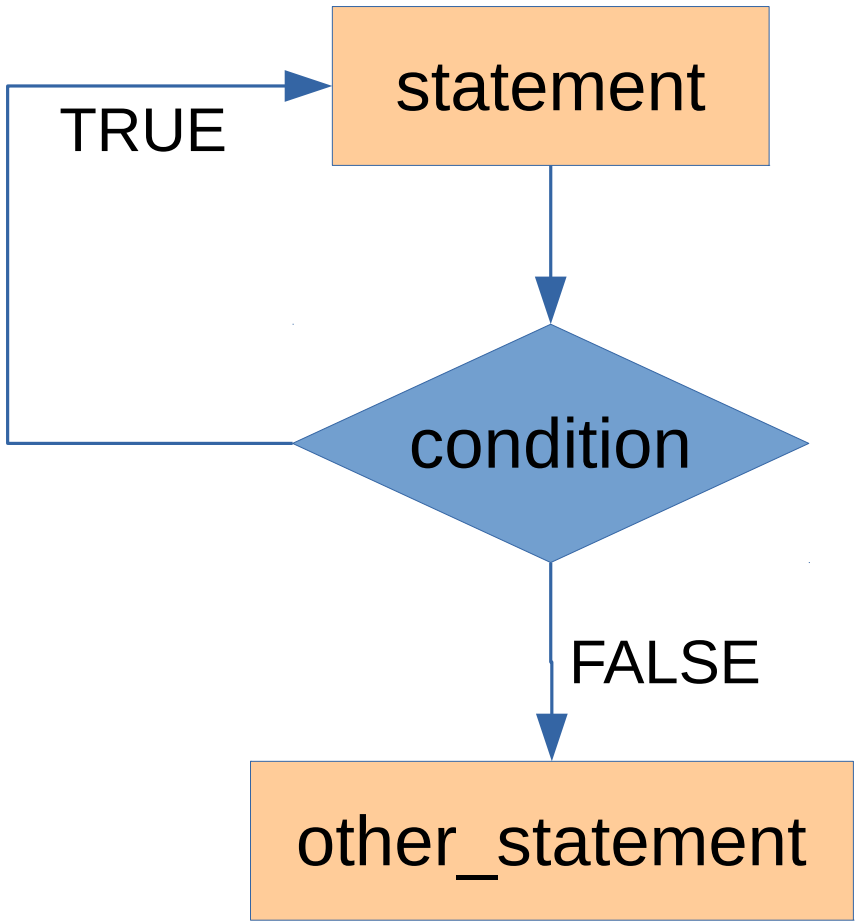
- General syntax:

```
1 do
2    statement;
3 while (condition);
```

```
1 do {
2    statement1;
3    statement2;
4 } while (condition);
```

- In this case the end condition is evaluated at the end
- The body is always executed at least once

`do ... while`: Flow Chart



do ... while: Example

```c
#include <stdio.h>
int main() {
  int n, sum = 0;
  do {
    printf("Enter number (<0 ends)");
    scanf("%d", &n);
    sum += n;
  } while (n >= 0);
  sum -= n; /* Remove last negative value */
  printf("The sum is %d\n", sum);
  return 0;
}
```

Jumping Out of a Cycle: `break`

- The keyword `break` allows to jump out of a cycle when executed
- We have already seen this while discussing `switch`

```
1 int num, i = 0;
2 scanf("%d", &num);
3 while (i < 50) {
4   printf("%d\n", i);
5   i++;
6   if (i == num)
7     break;
8 }
```

Jumping Out of a Cycle: `continue`

- `continue` jumps to the expression governing the cycle
- The expression is evaluated again and so on

```
1  char c;
2  /* code assumes that the input is
3     provided in one line like:
4     "abf23cdef" followed by enter */
5  while ((c = getchar()) != '\n') {
6    // ignore the letter b
7    if (c == 'b')
8      continue;
9    printf("%c", c);
10 }
```

Jumping Out of a Cycle

- Do not abuse `break` and `continue`
- You can always obtain the same result without using them
  - This at the price of longer coding
- By using them your code gets more difficult to read
- When you are experienced you will master their use
  - Meanwhile, learn the basics

Iterations: General Comments

- Inside the body of the loop you must insert an instruction that can cause the condition to become false
- If you do not do that, your program will fall into an infinite loop and will be unable to stop (Press `Ctrl-C` to stop such a program)
- `do ... while` is far less used than `while` and `for`
- The same constructs are provided in the majority of other programming languages

Arrays in C

- See first lecture for introduction
- In C you declare an array by specifying the size between square brackets
- Example: `int my_array[50];`
- The former is an array of 50 elements
- The first element is at position 0, the last one is at position 49

Accessing an Array in C

- To write an element, you specify its position

```
1   my_array[2] = 34;
2   my_array[0] = my_array[2];
```

- Pay attention: if you specify a position outside the limit, you will have unpredictable results segmentation fault, bus error, etc.
- And obviously wrong
- Note the different meaning of brackets
- Brackets in declaration describe the dimension, while in program they are the index operator

Arrays with Initialization

- C allows also the following declarations:

```
1   int first_array[]    = {12, 45, 7, 34};
2   int second_array[4] = {1, 4, 16, 64};
3   int third_array[4]   = {0, 0};
```

- It is not possible to specify more values than the declared size of the array
- The following is wrong:

```
1   int wrong[3] = {1, 2, 3, 4};
```

Typical Structure of a C Program

```c
#include <stdio.h>
int rect_area(int length, int width);
float b_func(int a, int b);
int main() {
  ...
  c = rect_area(5, 7);
  b_func(11, 6);
  return 0;
}
int rect_area(int length, int width) {
  ...    /* do some operations */
  return area;
}
float b_func(int a, int b) {
  ...    /* do some operations */
  return c;
}
```

Calling a Function

- To call a function you insert its name
  - Function call is a statement
- You have to provide suitable parameters
  - Number and type of parameters must match function declaration
- The result of a function can be ignored

An Example

```
1  #include <math.h>
2  #include <stdio.h>
3  int main() {
4    double number, root;
5    scanf("%lf", &number);
6    if (number >= 0) {
7      root = sqrt(number);
8      printf("Square root is %f\n", root);
9      sqrt(number); /* useless but legal */
10     /* What can I print now? */
11   }
12   else
13     printf("Cannot calc square root\n");
14   return 0;
15 }
```

```
gcc -Wall -lm -o example example.c
```