

CH-230-A

Programming in C and C++

C/C++

Tutorial 2

Dr. Kinga Lipskoch

Fall 2021

Type Conversions

- ▶ When data of different types are combined (via operators) some rules are applied
- ▶ Types are converted to a common type
 - ▶ Usually, to the larger one (called promotion)
 - ▶ **Example:** while summing an `int` and a `float`, the `int` is converted into a `float` and then the sum is performed
- ▶ A demotion is performed when a type is converted to a smaller one
 - ▶ **Example:** a function takes an `int` parameter and you provide a `float`
- ▶ A demotion implies possible loss of information
- ▶ Therefore, be careful with what to expect
 - ▶ In the above example, the fractional part will be lost

Casting

- ▶ It is possible to overcome standard conversions (casting)
- ▶ To force to a different data type, put the desired data type before the expression to be converted
(type name) expression
- ▶ Casting is a unary operator with high precedence

Casting: An Example

```
1      int a;  
2      float f1 = 3.456;  
3      float f2 = 1.22;  
4      /* these operations imply demotions */  
5      a = (int) f1 * f2;      /* a is now 3 */  
6      a = (int) (f1 * f2);   /* a is now 4 */
```

Incrementing and Decrementing

- ▶ The unary operators ++ and -- can be applied to increase or decrease a variable by 1

```
1  int a, b;  
2  a = b = 0;  
3  a++; b-- ; ++a ; --b;
```

- ▶ Note that they can be both **prefix** and **postfix** operators
 - ▶ The two versions are different

Prefix and Postfix Modes

- ▶ Prefix means that first you modify and then you use the value
- ▶ Postfix means that first you use and then you modify the value
- ▶ `int a = 10, b;`

Expression	New value of a	New value of b
<code>b = ++a;</code>	11	11
<code>b = a++;</code>	11	10
<code>b = --a;</code>	9	9
<code>b = a--;</code>	9	10

The `sizeof()` Operator

- ▶ `sizeof()` returns the number of bytes needed to store a specific object
- ▶ Useful for determining the sizes of the different data types on your system

```
1 int a;  
2 printf("size int %lu\n", sizeof(a));  
3 printf("size float %lu\n", sizeof(float));  
4 printf("size double %lu\n", sizeof(double));
```

- ▶ For strings do not confuse `sizeof()` with `strlen()`
- ▶ Compile-time operator, will not work for dynamically allocated memory

Boolean Variables

- ▶ A boolean variable can assume only two logic values: **true** or **false**
- ▶ Boolean variables and expressions are widely used in computer languages to control branching and looping
- ▶ Some operators return boolean values
- ▶ A boolean expression is an expression whose value is **true** or **false**

Boolean Operators

- ▶ Boolean operators can be applied to boolean variables
 - ▶ AND, OR, NOT

A	NOT A	A	B	A AND B	A	B	A OR B
false	true	false	false	false	false	false	false
true	false	false	true	false	false	true	true
		true	false	false	true	false	true
		true	true	true	true	true	true

Booleans in C

- ▶ Originally, C did not provide an ad-hoc boolean type but uses rather the `int` type
- ▶ 0 is false, everything different from 0 is true
- ▶ In C99 the type `_Bool` was introduced, **example:** `_Bool b = 0;`
- ▶ Additionally, the library `stdbool.h` defines the type `bool`, **example:** `bool b = false;`
- ▶ C also provides the three Boolean operators
 - ▶ `&&` for AND,
 - ▶ `||` for OR,
 - ▶ `!` for NOT
- ▶ Applied to booleans they return booleans

Boolean Operators: Example

```
1  int main() {
2      int a, b, c;
3      a = 0;                /* a is false */
4      b = 57;               /* b is true */
5      c = a || b;           /* c is true */
6      c = a && b;            /* c is false */
7      a = !a;               /* a is now true */
8      c = a && b;            /* c is now true */
9      c = (a && !b) && (a || b);
10     return 0;
11 }
```

Relational Operators

- ▶ Relational operators are applied to other data types (numeric, character, etc.) and produce boolean values
`(b > 5) --> true`
- ▶ Relational operators with boolean operators produce boolean expressions
`(b > 5) && (a < 1) --> true && false --> false`

Relational operator	Meaning
<code>==</code>	Equality test
<code>!=</code>	Inequality test
<code>></code>	Greater
<code><</code>	Smaller
<code>>=</code>	Greater or equal
<code><=</code>	Smaller or equal

Relational Operators: Example

```
1  int main() {  
2      int a = 2, b, c;  
3      float f1 = 1.34;  
4      float f2 = 3.56;  
5      char ch = 'D';  
6      b = f1 >= f2;  
7      c = !b;  
8      b = c == b;  
9      b = b != c;  
10     c = f2 > a;  
11     c = ch > a;  
12     return 0;  
13 }
```

Branching

- ▶ Up to now programs seem to execute all the instructions in sequence, from the first to the last (a **linear program**)
- ▶ Change the control flow of a program with **branching statements**
- ▶ Branching allows to execute (or not to execute) certain parts of a program depending on **boolean expressions** or **conditions**

Selection: `if ... else`

- ▶ In general selection constructs allow to choose a way in a binary bifurcation
- ▶ De facto you can use it in three ways
 - ▶ `if ()` single selection
 - ▶ `if ()`
 `else` double selection
 - ▶ `if ()`
 `else if ()`
 `else if ()`
 `...`
 `else` multiple selection

The if Syntax (1)

► General syntax:

```
1 if (condition)
2     statement 1;
3 else
4     statement 2;
5 other_statement; /* always executed */
```

- The else part can be omitted
- Statement: single statement or multiple statements
- Multiple statements need to be surrounded by braces { }

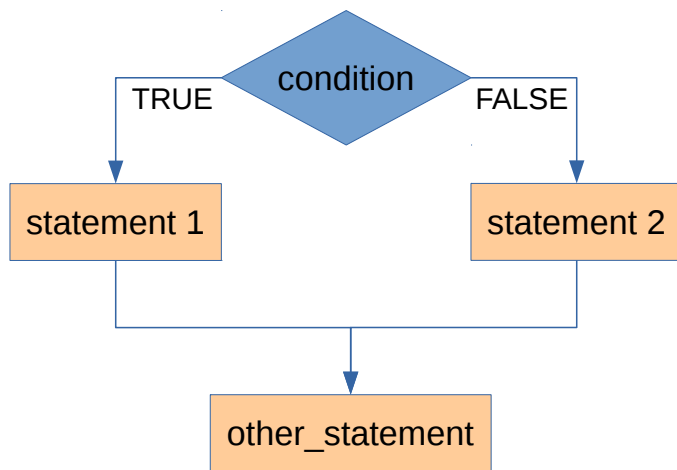
The `if` Syntax (2)

- ▶ Preferred syntax (always use braces)

```
1 if (condition) {  
2     statements;  
3 }  
4 else {  
5     statements;  
6 }
```

- ▶ If you add statements, program flow is not changed (less errors)
- ▶ Using indentation, you can easily see where block starts and ends

if: Flow Chart



if: Example

```
1 #include <stdio.h>
2 int main() {
3     int first, second;
4     printf("Type the first number:\n");
5     scanf("%d", &first);
6     printf("Type the second number:\n");
7     scanf("%d", &second);
8     if (first > second) {
9         printf("The larger one is %d\n", first);
10    }
11    else {
12        printf("The larger one is %d\n", second);
13    }
14    printf("Can you see the logical error?\n");
15    return 0;
16 }
```

Statements and Compound Statements

- ▶ Statements can be grouped together to form compound statements
- ▶ A compound statement is a set of statements surrounded by braces

```
1 int a = 3;
2 if (a > 0) {
3     printf("a is positive %d\n", a);
4     a = a - 2 * a;
5     printf("now a is negative %d\n", a)
6 }
```