Informatik I - Exercise Session
Variable Types, Expressions, Loops and Scopes

Variable Types

- `int, unsigned int`
- `bool`
- `float, double`
- `... // more to come`

Do you know the “ranking” of these types when converting one to another?

Variable Types: Conversion Ranking

This is very important to keep in mind when writing complex expressions involving conversion of one of these (numeric) types into another:

`bool` < `int` < `unsigned int` < `float` < `double`

Variable Types: Conversion Ranking

This is very important to keep in mind when writing complex expressions involving conversion of one of these (numeric) types into another:

`bool < int < unsigned int < float < double`

IMPORTANT: `unsigned int` is "bigger" or more important than `int`, since it contains more possible positive values and is thus preferred in calculations.

Variable Types: Conversion Ranking

This is very important to keep in mind when writing complex expressions involving conversion of one of these (numeric) types into another:

```
bool < int < unsigned int < float < double
```

IMPORTANT: `unsigned int` is "bigger" or more important than `int`, since it contains more possible positive values and is thus preferred in calculations.

For the "non-standard" types of `unsigned int` and `float`, there are suffixes to explicitly return these types in literals:

```
?? i = 3;    // int
?? j = 3u;   // unsigned int
?? k = 2.6;  // double
?? l = 2.6f; // float
```

Exercise I

1. Which of the following character sequences are not C++ expressions, and why not? Here, `x` and `y` are variables of type `int`.
   a) `(y++ < 0 && y < 0) + 2.0`
   b) `y = (x++ = 3)`
   c) `3.0 + 3 - 4 + 5`
   d) `5 % 4 * 3.0 + true * x++`
2. For all of the valid expressions that you have identified in 1, decide whether these are lvalues or rvalues, and explain your decisions.
3. Determine the values of the expressions and explain how these values are obtained. Assume that initially `x == 1` and `y == -1`.

Exercise I: Solution 1)

```
(y++ < 0 && y < 0) + 2.0
```

Exercise I: Solution 1)

```
(y++ < 0 && y < 0) + 2.0

    (-1 < 0 && y < 0) + 2.0  // after this step: y==0
```

Exercise I: Solution 1)

```
(y++ < 0 && y < 0) + 2.0

    (-1 < 0 && y < 0) + 2.0  // after this step: y==0
    (true && y < 0) + 2.0
```

Exercise I: Solution 1)

```
(y++ < 0 && y < 0) + 2.0

    (-1 < 0 && y < 0) + 2.0  // after this step: y==0
    (true && y < 0) + 2.0
    (true && false) + 2.0
```

Exercise I: Solution 1)

```
(y++ < 0 && y < 0) + 2.0

    (-1 < 0 && y < 0) + 2.0   // after this step: y==0
    (true && y < 0) + 2.0
    (true && false) + 2.0
    (false) + 2.0
```

Exercise I: Solution 1)

```
(y++ < 0 && y < 0) + 2.0

    (-1 < 0 && y < 0) + 2.0   // after this step: y==0
    (true && y < 0) + 2.0
    (true && false) + 2.0
    (false) + 2.0
    0.0 + 2.0
```

Exercise I: Solution 1)

```
(y++ < 0 && y < 0) + 2.0

    (-1 < 0 && y < 0) + 2.0  // after this step: y==0
    (true && y < 0) + 2.0
    (true && false) + 2.0
    (false) + 2.0
    0.0 + 2.0
    2.0
```

Exercise I: Solution 1)

```
(y++ < 0 && y < 0) + 2.0

    (-1 < 0 && y < 0) + 2.0  // after this step: y==0
    (true && y < 0) + 2.0
    (true && false) + 2.0
    (false) + 2.0
    0.0 + 2.0
    2.0
```

Exercise I: Solution 1)

```
(y++ < 0 && y < 0) + 2.0
```

```
(-1 < 0 && y < 0) + 2.0  // after this step: y==0
(true && y < 0) + 2.0
(true && false) + 2.0
(false) + 2.0
0.0 + 2.0
2.0
```

R-VALUE

Exercise I: Solution 2)

```
y = (x++ = 3)
```

Exercise I: Solution 2)

```
y = (x++ = 3)
```

INVALID

Exercise I: Solution 3)

```
3.0 + 3 - 4 + 5
```

Exercise I: Solution 3)

```
3.0 + 3 - 4 + 5
```

```
((3.0 + 3) - 4) + 5
```

Exercise I: Solution 3)

```
3.0 + 3 - 4 + 5
```

```
((3.0 + 3) - 4) + 5
((3.0 + 3.0) - 4) + 5
```

Exercise I: Solution 3)

```
3.0 + 3 - 4 + 5
```

```
((3.0 + 3) - 4) + 5
((3.0 + 3.0) - 4) + 5
(6.0 - 4) + 5
```

Exercise I: Solution 3)

```
3.0 + 3 - 4 + 5

    ((3.0 + 3) - 4) + 5
    ((3.0 + 3.0) - 4) + 5
    (6.0 - 4) + 5
    (6.0 - 4.0) + 5
```

Exercise I: Solution 3)

```
3.0 + 3 - 4 + 5
```

```
((3.0 + 3) - 4) + 5
((3.0 + 3.0) - 4) + 5
(6.0 - 4) + 5
(6.0 - 4.0) + 5
2.0 + 5
```

Exercise I: Solution 3)

```
3.0 + 3 - 4 + 5
```

```
((3.0 + 3) - 4) + 5
((3.0 + 3.0) - 4) + 5
(6.0 - 4) + 5
(6.0 - 4.0) + 5
2.0 + 5
2.0 + 5.0
```

Exercise I: Solution 3)

```
3.0 + 3 - 4 + 5
```

```
((3.0 + 3) - 4) + 5
((3.0 + 3.0) - 4) + 5
(6.0 - 4) + 5
(6.0 - 4.0) + 5
2.0 + 5
2.0 + 5.0
7.0
```

Exercise I: Solution 3)

```
3.0 + 3 - 4 + 5
```

```
((3.0 + 3) - 4) + 5
((3.0 + 3.0) - 4) + 5
(6.0 - 4) + 5
(6.0 - 4.0) + 5
2.0 + 5
2.0 + 5.0
7.0
```

R-VALUE

Exercise I: Solution 4)

```
5 % 4 * 3.0 + true * x++
```

Exercise I: Solution 4)

```
5 % 4 * 3.0 + true * x++
```

```
((5 % 4) * 3.0) + (true * (x++))
```

Exercise I: Solution 4)

```
5 % 4 * 3.0 + true * x++
```

```
((5 % 4) * 3.0) + (true * (x++))
(1 * 3.0) + (true * (x++))
```

Exercise I: Solution 4)

```
5 % 4 * 3.0 + true * x++
```

```
((5 % 4) * 3.0) + (true * (x++))
(1 * 3.0) + (true * (x++))
(1.0 * 3.0) + (true * (x++))
```

Exercise I: Solution 4)

```
5 % 4 * 3.0 + true * x++
```

```
((5 % 4) * 3.0) + (true * (x++))
(1 * 3.0) + (true * (x++))
(1.0 * 3.0) + (true * (x++))
3.0 + (true * (x++))
```

Exercise I: Solution 4)

```
5 % 4 * 3.0 + true * x++
```

```
((5 % 4) * 3.0) + (true * (x++))
(1 * 3.0) + (true * (x++))
(1.0 * 3.0) + (true * (x++))
3.0 + (true * (x++))
3.0 + (true * 1)
```

Exercise I: Solution 4)

```
5 % 4 * 3.0 + true * x++
```

```
((5 % 4) * 3.0) + (true * (x++))
(1 * 3.0) + (true * (x++))
(1.0 * 3.0) + (true * (x++))
3.0 + (true * (x++))
3.0 + (true * 1)
3.0 + (1 * 1)
```

Exercise I: Solution 4)

```
5 % 4 * 3.0 + true * x++
```

```
((5 % 4) * 3.0) + (true * (x++))
(1 * 3.0) + (true * (x++))
(1.0 * 3.0) + (true * (x++))
3.0 + (true * (x++))
3.0 + (true * 1)
3.0 + (1 * 1)
3.0 + 1
```

Exercise I: Solution 4)

```
5 % 4 * 3.0 + true * x++
```

```
((5 % 4) * 3.0) + (true * (x++))
(1 * 3.0) + (true * (x++))
(1.0 * 3.0) + (true * (x++))
3.0 + (true * (x++))
3.0 + (true * 1)
3.0 + (1 * 1)
3.0 + 1
3.0 + 1.0
```

Exercise I: Solution 4)

```
5 % 4 * 3.0 + true * x++
```

```
((5 % 4) * 3.0) + (true * (x++))
(1 * 3.0) + (true * (x++))
(1.0 * 3.0) + (true * (x++))
3.0 + (true * (x++))
3.0 + (true * 1)
3.0 + (1 * 1)
3.0 + 1
3.0 + 1.0
4.0
```

Exercise I: Solution 4)

```
5 % 4 * 3.0 + true * x++
```

```
((5 % 4) * 3.0) + (true * (x++))
(1 * 3.0) + (true * (x++))
(1.0 * 3.0) + (true * (x++))
3.0 + (true * (x++))
3.0 + (true * 1)
3.0 + (1 * 1)
3.0 + 1
3.0 + 1.0
4.0
```

R-VALUE

Loop Correctness

Can a user of the program observe the difference between the output produced by these three loops? If yes, how? Assume that `n` is a variable of type `int` whose value is given by the user.

```
int n; std::cin >> n;
int i;

// loop 1
for (i = 1; i <= n; ++i) {
  std::cout << i << "\n";
}

// loop 2
i = 0;
while (i < n) {
    std::cout << ++i << "\n";
}
```

```
// loop 3
i = 1;
do {
  std::cout << i++ << "\n";
} while (i <= n);
```

Loop Correctness - Solution

There are the following differences:

- Unlike loops 1 and 2, loop 3 does output 1 for input `n == 0` because the statement in a `do`-loop is always executed once, before the condition is checked.
- If n is the largest possible positive integer, then the loops 1 and 3 exhibit undefined behavior because `++i` increases `i` beyond the maximum integer value before the condition `i <= n` can stop the loop.

Loop Conversion

Convert the following `for`-loop into an equivalent `while`-loop:

```
for (int i = 0; i < n; ++i)
    BODY
```

Convert the following `while`-loop into an equivalent `for`-loop:

```
while (condition)
    BODY
```

Convert the following `do`-loop into an equivalent `for`-loop:

```
do
    BODY
while (condition);
```

Loop Conversion - Solution

A possible way to convert a `for`-loop into an equivalent `while`-loop:

Loop Conversion - Solution

A possible way to convert a `for`-loop into an equivalent `while`-loop:

```
{   // This additional block restricts the scope of i.
  int i = 0;
  while (i < n) {
    BODY
    ++i;
  }
}
```

Loop Conversion - Solution

A possible way to convert a `for`-loop into an equivalent `while`-loop:

```
{   // This additional block restricts the scope of i.
  int i = 0;
  while (i < n) {
    BODY
    ++i;
  }
}
```

A possible way to convert a `while`-loop into an equivalent `for`-loop:

Loop Conversion - Solution

A possible way to convert a `for`-loop into an equivalent `while`-loop:

```
{   // This additional block restricts the scope of i.
  int i = 0;
  while (i < n) {
    BODY
    ++i;
  }
}
```

A possible way to convert a `while`-loop into an equivalent `for`-loop:

```
for ( ;condition; )
  BODY
```

Loop Conversion - Solution

A possible way to convert a `for`-loop into an equivalent `while`-loop:

```
{   // This additional block restricts the scope of i.
  int i = 0;
  while (i < n) {
    BODY
    ++i;
  }
}
```

A possible way to convert a `while`-loop into an equivalent `for`-loop:

```
for ( ;condition; )
  BODY
```

A possible way to convert a `do`-loop into an equivalent `for`-loop:

Loop Conversion - Solution

A possible way to convert a `for`-loop into an equivalent `while`-loop:

```
{    // This additional block restricts the scope of i.
  int i = 0;
  while (i < n) {
    BODY
    ++i;
  }
}
```

A possible way to convert a `while`-loop into an equivalent `for`-loop:

```
for ( ;condition; )
  BODY
```

A possible way to convert a `do`-loop into an equivalent `for`-loop:

```
BODY
for ( ;condition; )
  BODY
```

Scopes

What is a scope in a C++ program? What does it do?

Scopes

What is a scope in a C++ program? What does it do?

Answer: Scopes define the code segments of our program in which a variable (lvalue) exists. The scope of a variable starts at the point of its definition and ends at the end of the block where it was defined. The following example does not work:

```cpp
if (x < 7) {
    int a = 8;
    std::cout << a; // Fine, prints 8.
}
std::cout << a; // Compiler error, a does not exist.
```

How would we fix this error?

Scopes - Example I

One possibility to fix this would be:

```
int a = 2;
if (x < 7) {
    int a = 8;
    std::cout << a; // Fine, prints 8.
}
std::cout << a; // Prints 2. Reason: scopes.
```

What does this print? And why?

Scopes - Example I

One possibility to fix this would be:

```
int a = 2;
if (x < 7) {
    int a = 8;
    std::cout << a; // Fine, prints 8.
}
std::cout << a; // Prints 2. Reason: scopes.
```

What does this print? And why?

Bad programming style, don't do this. There is always another way to name your variables.

Scopes - Example II

What is the scope of `sum`, `i`, and `a` in the following snippet?

```
int sum = 0;
for (int i = 0; i < 5; ++i) {
    int a;
    std::cin >> a;
    sum += a;
}
```

Scopes - Example II

What is the scope of `sum`, `i`, and `a` in the following snippet?

```
1 int sum = 0;
2 for (int i = 0; i < 5; ++i) {
3     int a;
4     std::cin >> a;
5     sum += a;
6 }
```

- `sum`: From line 1 to at least after line 6 (and possibly more)
- `i`: The entire `for`-loop
- `a`: Only one loop iteration

Rewrite this exact loop using `while`.

Scopes - Example II

```
int sum = 0;
{
    int i = 0;
    while (i < 5) {
        int a;
        std::cin >> a;
        sum += a;
        ++i;
    }
}
```

This does not work every time, but most simple loops (without `break` or `continue`) can be rewritten this way to achieve the same scopes.