# 2. Variables and Conditions

Ján Dugáček

September 25, 2019

# Table of Contents

**Variables**
Conditions
Homework

Why we need them
Available types
Usage
Exercises
Shortcuts

## Advanced exercise

- Do this only if you already know how to use variables!
- Calculate $\pi$ using the Monte Carlo method (scatter many points randomly in a square, calculate the fraction of them that is closer to its centre than a half of the square's side)
- Hint: you may use `rand()` to generate random numbers
- Why is the result so imprecise?
- Challenge: Do it without computing any square root (neither manually nor in the program)
- Second powers of the same numbers are computed over and over. Would it be useful to store the computed second powers of numbers for later use?

**Variables**
Conditions
Homework

Why we need them
Available types
Usage
Exercises
Shortcuts

# Variables

- *Everything* in digital format is a number or a group of numbers (addresses, texts, pictures, programs, ...)
- There are several formats for numbers, depending on the required size and need to support negative numbers and decimals
- Numbers are always binary code, groups of ones and zeroes, a bit is a single value that can be zero or one, a byte is a group of eight bits ($8^2 = 256$ possible values)
- On computers, numbers usually can be saved on 1 byte (256 values), 2 bytes ($2^{16} = 65536$ values), 4 bytes ($2^{32} = 4294967296$ values) or 8 bytes ($2^{64} = 18446744073709551616 = 1.8 \cdot 10^{19}$ values)
- A number stored someplace with a name is called *variable*
- A single number is called *primitive data type*

**Variables**
Conditions
Homework

Why we need them
Available types
Usage
Exercises
Shortcuts

# Usage

```cpp
#include <iostream>
int main(int argc, char** argv) {
        int x;
        x = 2;
        std::cout << x << std::endl;
        return 0;
}
```

- We first create variable x
- The compiler will recognise x as an integer variable
- Then we set value 2 to x
- We can write its value to the program's output

# Usage #2

```
#include <iostream>
int main(int argc, char** argv) {
        int x = 2;
        std::cout << x << std::endl;
        return 0;
}
```

- We can set its value at the same line as when creating it
- This is the recommended way to do it, because if you forget to set it, it will have an unpredictable value

Variables
Conditions
Homework

Why we need them
Available types
Usage
Exercises
Shortcuts

# Available types

- `int` - standard sized integer (usually `int32_t`, range -2147483648 to 2147483647)
- `short int` - short sized integer (usually `int16_t`, range -32768 to 32767)
- `char` - very short sized integer, often used to store letters (usually `int8_t`, range -128 to 127)
- `long int` - short sized integer (usually `int64_t`, range -9223372036854775808 to 9223372036854775807)
- `unsigned int` - integer for non-negative values (usually `uint32_t`, range 0 to 4294967295)
- There are unsigned versions of all other sized integer types

Variables
Conditions
Homework

Why we need them
Available types
Usage
Exercises
Shortcuts

## Available types

- `float` - stores numbers with decimal point (usually 32-bit, 6 decimals, greatest numbers are around $10^{38}$)
- `double` - stores numbers with decimal point (usually 64-bit, 15 decimals, greatest numbers are around $10^{308}$)
- `bool` - can have only two values, `false` which is 0 or `true` which is 1
- `std::string` - stores text, works quite differently

# Usage #3

```cpp
#include <iostream>
int main(int argc, char** argv) {
        int x = -1024 - 2;
        short int y = x * x;
        int z = x / 4;
        std::cout << y << std::endl;
        std::cout << z << std::endl;
        return 0;
}
```

- We first create variable x and save -1026 into it
- Then we create variable y and save the square of x into it, which does not fit there
- After, we create variable z and set its value to x divided by 4, because both x and 4 are integers, the result is an integer, rounding the value down
- The resulting values of y and z are written into the terminal

# Usage #3

```
#include <iostream>
int main(int argc, char** argv) {
        float x = 15 / 2;
        float y = 15.0 / 2;
        float z = (float)15 / 2;
        float w = x / 2;
        std::cout << "Computed x=" << x << " y= " << y
                << " z= " << z " w= " << w << std::endl;
        return 0;
}
```

- We first divide 15 by 2, rounding down because both numbers are integers and result is integer, recalculate it to float and save it into x
- Then we divide 15.0 by 2, because 15.0 is a decimal, it is a float, arithmetic between a float and an int yields a float, the resulting float is saved into y
- After, we convert the integer 15 to float, divide it by 2, the resulting float is saved into z
- Next, we divide the float x by 2 and save it into variable w
- The resulting values of variables are written into the terminal

**Variables**
Conditions
Homework

Why we need them
Available types
Usage
**Exercises**
Shortcuts

## Exercises

1. Set 17 to x, divide it by 4 (rounded down), set $x^2 - 12$ to y, add 18 to the result and write out the result

2. Calculate
$(3 + 2 - 12) \cdot ((9 - 2) \cdot 5) + (3 + 2 - 12) \cdot (8 + ((9 - 2) \cdot 5))$
without writing 3 + 2 – 12 or (9 – 2) · 5 more than once or calculating anything yourself

3. Calculate $\frac{3+2-12}{(9-2)\cdot 5} + (3 + 2 - 12) \cdot (8 + \frac{(9-2)\cdot 5}{3+2-12})$ without writing 3 + 2 – 12 or (9 – 2) · 5 more than once or calculating anything yourself

**Variables**
Conditions
Homework

Why we need them
Available types
Usage
Exercises
**Shortcuts**

# Shortcuts

- Lines like x = x + 4 are used a lot, so they can be shortened to x += 4
- Analogically, you can use x -= y * 2 (subtract 2 multiplied by y from x and save it into x), x /= 1.5 or x *= 1.01
- x += 1 can be further shortened to x++ or ++x
- Analogically, there is also x-- or --x for x -= 1

## Advanced exercise

- Do this only if you already know how to use `if`, `while` and `for`!
- Calculate $x$ in $x + 1 = \frac{1}{x}$
- You may assume that $x$ is positive
- Challenge: Do not calculate anything more than 1000 times, but limit your precision only by the maximum decimals that can be stored in primitive types and use no prior knowledge

# Condition

```
int x;
std::cin >> x;
if (x < 0)
        x *= -1;
std::cout << x << std::endl;
```

- First, we let the user insert a number
- Then, we check if x is lesser than 0
- Only if x is lesser than 0, multiply by -1
- This will replace x by its absolute
- x is printed at the end of the program

# Condition #2

```
bool changed = false;
if (x >= 0) {
        x *= -1;
        changed = true;
}
```

- Here, we check if x is greater than or equal to 0
- If the condition is met, multiply x by -1 and set variable changed to 1
- Variables defined in a *block* (the part in curly brackets) are not available outside of it

# Condition #3

```
int changed = 0;
bool equals = (x == y);
if (x > y || x < 2 * y) {
        changed = 1;
        if (equals) {
                changed = 2;
        }
}
```

- Here, we check if x is greater than y *or* x is less than two times y
- We also check if x equals y and save the result of the comparison into variable equals
- If the first condition is met, 1 is assigned to changed and we check if x was *previously found to be* equal to y
- The result of comparison can be 1 (true) or 0 (false)

# Condition #4

```
int z = 0;
if (x > y && x != 1) {
        z = 1;
        if (x = y - 1) {
                z = 2;
        }
}
```

- Here, we check if x is greater than y *and* x is not equal to 1
- If the condition is met, 1 is assigned to z and y - 1 is assigned to x and *if x is non-zero (true)*, 2 is assigned to z
- **Do not confuse = (variable assignment) with == (comparison)! It is a huge source of errors!**

# Condition #5

```
int z = 0;
if (x > y && (x = y || y == 1)) {
    z = 1;
}
```

- Here, we check if x is greater than y *and if that is true*, we assign y into x and if the result is non-zero (true) or y is equal to 1, the condition is met
- If the condition is met, 1 is assigned to z
- If x is not greater than y, the condition is never true and the rest is ignored, thus y is never assigned to x
- Do not confuse && and || with & and |, they mean something else but usually lead to different outcomes, so a program using & instead of && may seem okay but then behave weirdly

# Condition #6

```
int z = 0;
if (!(x > y) && (x == 1 || (x = y))) {
        z = 1;
}
```

- Here, we check if it's not true that x is greater than y *and if that condition is met*, we check if x is equal to one, *if that is false*, we assign y into x, check if it's non-zero and go inside the block if the one of these two conditions is met
- If x is equal to 1, the condition is true regardless of the value of y and the next condition is ignored, thus y is never assigned to x

## Inline condition

```
int z = (!( x > y) && ( x == 1 || ( x = y ))) ? 1 : 0;
```

- This does the same as the previous, if the condition is met, z is initialised with 1, otherwise it's initialised with 0
- It is useful only when assigning values into a variable depending on a condition

```
int z = ( x > 1) ? (( y > 1) ? 2 : 1) : 0;
```

- It can be nested too
- If x is greater than 1, then if y is greater than 1, 2 is set into z, otherwise 1, if x is not greater than 1, 0 is set into z

# Exercise

1. Create a program that reads a number and tells if it's even or odd
2. Create a program that reads a number and reports if it's the square of an integer (the number will not be greater than 20)
3. Create a program that reads two numbers as coordinates of a point and prints the point's distance from point (2, 3)
4. Create a program that reads two numbers as coordinates of a point and determines if the point lies within a circle with centre at (2, 3) and radius 4

# Homework

- No homework