Lesson 3

Introduction to C++

Operators is C++ (Summary)

Precedence	Operator	Description	Associativity
1	::	Scope resolution operator (four dots)	left to right
2	() ++ -- 3 hidden	Operator function call Post-increment Post-decrement .	left to right
3	! ~ ++ -- - + (type) sizeof 2 hidden	Logical negation Bitwise complement Pre-increment Pre-decrement Unary minus Unary plus Cast to a given type Return size in bytes .	right to left
4	2 hidden		left to right
5	* / %	Multiplication Division Modulus	left to right
6	+ -	Addition Subtraction	left to right
7	<< >>	Bitwise shift left Bitwise shift right	left to right
8	< <= > >=	Comparison less-than Comparison less-than-or-equal-to Comparison greater-than Comparison geater-than-or-equal-to	left to right
9	== !=	Comparison equal-to Comparison not-equal-to	left to right
10	&	Bitwise AND	left to right
11	^	Bitwise exclusive OR	left to right
12	|	Bitwise inclusive (normal) OR	left to right
13	&&	Logical AND	left to right
14	||	Logical OR	left to right
15	? :	Ternary conditional (if-then-else)	right to left
16	= += -= *= /= %= &= ^= |= <<= >>=	Assignment operator Increment and assign Decrement and assign Multiply and assign Divide and assign Modulo and assign Bitwise AND and assign Bitwise exclusive OR and assign Bitwise inclusive (normal) OR and assign Bitwise shift left and assign Bitwise shift right and assign	right to left
17	,	Sequential evaluation operator	left to right

What are the values of x and y equal to after execution of this instruction (by default x and y equal to 13):

x += y %= (3, 4, 5 * 1 + 2 * 4 - 3)

Control sequences

if condition

Syntax :

if (**boolean expression or expression, convertible to bool**)
    statement or composite-statement

Statement will be executed if and only if the expression in parenthesis evaluates to true

```cpp
int main () {
    int x;
    std::cout << "Input x" << std::endl;
    std::cin >> x;

    if (x % 3 != 0)
        std::cout << "x is not divisible by 3" << std::endl;

    if (x % 2 != 0)
        std::cout << "x is not divisible by 2" << std::endl;

    return 0;
}
```

```
Input x: 123
x is not divisible by 2
```

```
Input x: 5
x is not divisible by 3
x is not divisible by 2
```

```
Input x: 666
```

Control sequences

else if condition

Syntax :
Else if statent must follow if statement or else if statement. Otherwise, the program won't compile.

else if (**boolean expression, or expression, convertible to bool**)
    statement or composite-statement

Statement will be executed if and only if the expression in parenthesis evaluates to true and the expression in parenthesis of the previos if condition evaluates to false and all of the expressions in parenthesis of previous else if conditions also evluates to false

```
int main () {
    int x;
    std::cout << "Input x: ";
    std::cin >> x;

    if (x % 3 == 0)
        std::cout << "x divisible by 3" << std::endl;

    else if (x % 3 == 1)
        std::cout << "x mod 3 = 1" << std::endl;

    else if (x % 3 == 2)
        std::cout << "x mod 3 = 2" << std::endl;

    return 0;
}
```

```
Input x: 123
x divisible by 3
```

```
Input x: 5
x mod 3 = 2
```

```
Input x: 4
x mod 3 = 1
```

Control sequences

else condition

Syntax:
Else statent must follow if condition or else if condition. Otherwise, the program won't compile.

else
    statement or composite-statement

Statement will be executed if and only if the expression in parenthesis of the previos if condition evaluates to false and all of the expressions in parenthesis of previous else if conditions also evluates to false

```cpp
int main () {
    int x;
    std::cout << "Input x: ";
    std::cin >> x;

    if (x % 3 == 0)
        std::cout << "x divisible by 3" << std::endl;

    else if (x % 3 == 1)
        std::cout << "x mod 3 = 1" << std::endl;

    else
        std::cout << "x is not divisible by 3 and x mod 3 =/= 1" << std::endl;

    return 0;
}
```

```
Input x: 123
x divisible by 3
```

```
Input x: 124
x mod 3 = 1
```

```
Input x: 5
x is not divisible by 3 and x mod 3 =/= 1
```

Control sequences

while loop

Syntax:

while (**boolean expression, or expression, convertible to bool**)
    statement or composite-statement

Statement or composite statement will be executed over and over again as long as the condition is true.

```cpp
int main () {
    int x;
    std::cout << "Input x: ";
    std::cin >> x;

    while (x % 7)
        std::cout << "x still is not divisible by 7. Current x: " << x -- << std::endl;

    std::cout << "Now x is divisible by 7: x = " << x << std::endl;

    return 0;
}
```

```
Input x: 123
x still is not divisible by 7. Current x: 123
x still is not divisible by 7. Current x: 122
x still is not divisible by 7. Current x: 121
x still is not divisible by 7. Current x: 120
Now x is divisible by 7: x = 119
```

Control sequences

while loop

Syntax:

while (**boolean expression, or expression, convertible to bool**)
    statement or composite-statement

Statement or composite statement will be executed over and over again as long as the condition is true.

```cpp
int main () {
    int x;
    std::cout << "Input x: ";
    std::cin >> x;

    while (x % 7)
        std::cout << "x still is not divisible by 7. Current x: " << x -- << std::endl;

    std::cout << "Now x is divisible by 7: x = " << x << std::endl;

    return 0;
}
```

```
Input x: 123
x still is not divisible by 7. Current x: 123
x still is not divisible by 7. Current x: 122
x still is not divisible by 7. Current x: 121
x still is not divisible by 7. Current x: 120
Now x is divisible by 7: x = 119
```

Control sequences

do-while loop

Syntax:

do
    statement or composite-statement
while (**boolean expression, or expression, convertible to bool**);

Statement or composite statement will be executed unconditionally once, then, if the expression in condition evaluates to true, the statement will be executed again, etc...

```
int main () {
    int x;
    std::cout << "Input x: ";
    std::cin >> x;

    do
        std::cout << x << std::endl;
    while (-- x);

    return 0;
}
```

```
Input x: 3
3
2
1
```

Control sequences

for loop

Syntax:

for (**declaration or expr [init statement]**; **boolean expression [condition]**; **expression [iter expr]**)
    statement or composite statement

The above syntax produces code almost equivalent to:

```
{
    init-statement
    while ( condition ) {
        statement
        iteration-expression ;
    }
}
```

Exaple:

```
int main () {
    for (int i = 1; i < 9; i += 2)
        std::cout << i << " ";

    return 0;
}
```

```
1 3 5 7
```

Control sequences

for loop

Every instruction in parenthesis can be omitted.
Also, even statement can be omitted (same as for the while loop)

Infinite for loop:

```cpp
int main () {
    for (;;)
        std::cout << "Hi!" << std::endl;

    return 0;
}
```

Omitted statement:

```cpp
int main () {
    int x = 3;
    for (; x --; std::cout << "Hi!" << std::endl);

    return 0;
}
```

```
Hi!
Hi!
Hi!
```

Omitted statement:

```cpp
int main () {
    int x = 3;
    while (std::cout << x << " ", -- x);

    return 0;
}
```

```
3 2 1
```

Control sequences

Continue and break keywords

Continue
Unconditionally proceeds to the next iteration of the current loop

Break
Unconditionally terminates the current loop

Important:
break is not the same as return
(lol, how can you possibly confuse them… ?)

```cpp
int main () {
    int n;
    std::cin >> n;

    for (int i = 0; i < 5; i ++) {
        if (i == 6)
            break;

        if (i % 2)
            continue;

        std::cout << i << std::endl;
    }

    return 0;
}
```

```
10
0
2
4
```

Control sequences

Composite statement examples:

```cpp
int main () {
    int x;
    std::cin >> x;

    if (x == 0) {
        std::cout << "First line" << std::endl;
        std::cout << "Second line" << std::endl;
    }

    return 0;
}
```

```
0
First line
Second line
```

Control sequences

Common mistake:

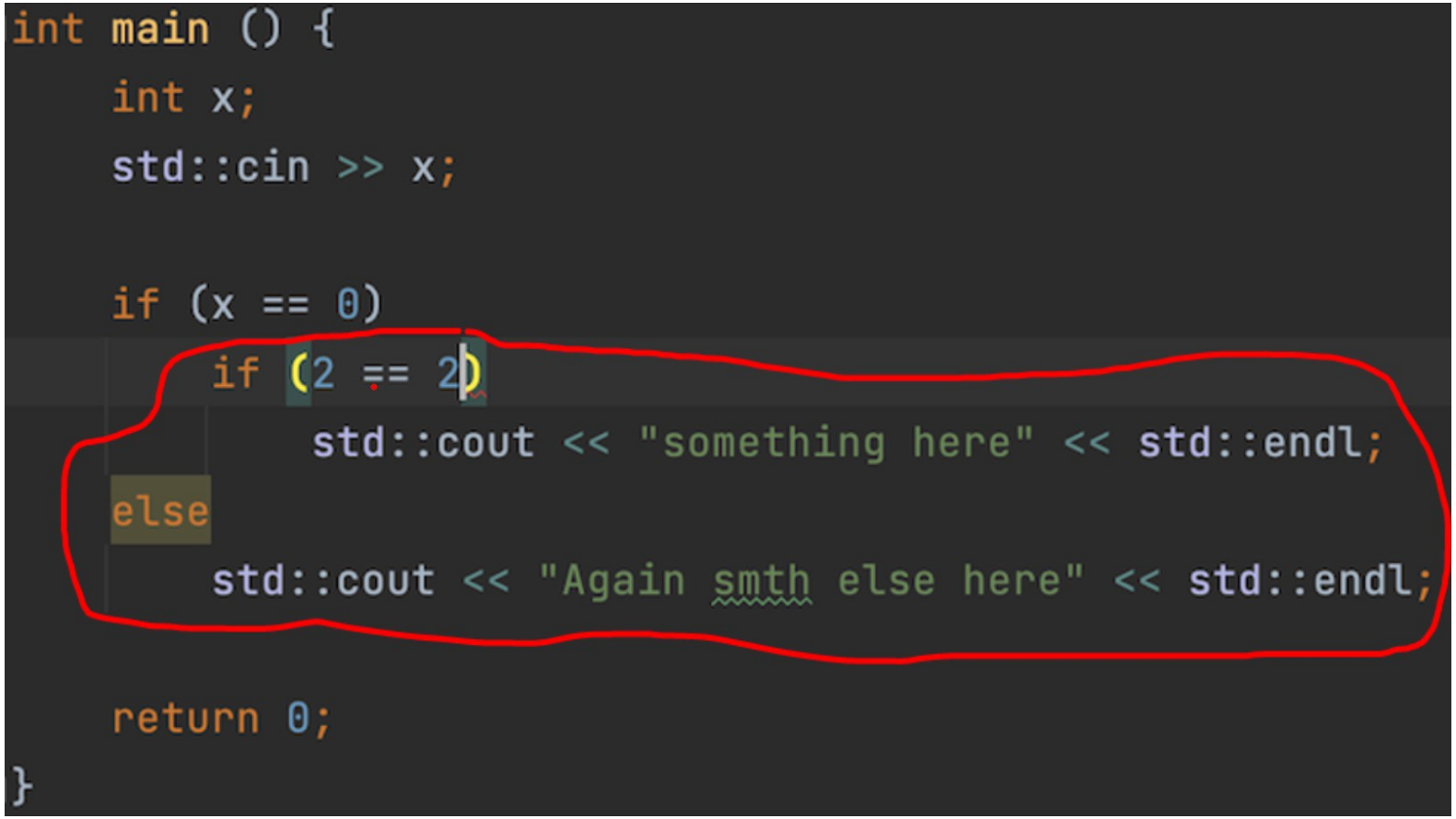
```
int main () {
    int x;
    std::cin >> x;

    if (x == 0)
        if (2 == 2)
            std::cout << "something here" << std::endl;
    else
        std::cout << "Again smth else here" << std::endl;

    return 0;
}
```

Control sequences

Common mistake:



Control sequences

Solution:

```cpp
int main () {
    int x;
    std::cin >> x;

    if (x == 0) {
        if (2 == 2)
            std::cout << "something here" << std::endl;
    }

    else
        std::cout << "Again smth else here" << std::endl;

    return 0;
}
```

```
2
Again smth else here
```

```
3
Again smth else here
```

Control sequences

How to avoid:

BAD: Always put figure brackets and use composite statements…

```cpp
int main () {
    int x;
    std::cin >> x;

    if (true) {
        std::cout << "Only one line here...";
    }

    if (true) {
        std::cout << "Again only one line here...";
    }

    return 0;
}
```

GOOD: Memorize how this mechanism works

Control sequences

Switch / case

Nice replacement for if-else if-else when you have similar conditions in parenthesis:

```
switch (expression)
 { case expression:
        statements
        [break;]
    …
    default:
        statements
        [break;]
}
```

Control sequences

Switch / case

Example:

```cpp
int main () {
    int n;
    std::cin >> n;

    if (n % 5 == 0)
        std::cout << "n is divisible by 5" << std::endl;

    else if (n % 5 == 1 || n % 5 == 3)
        std::cout << "n mod 5 is odd" << std::endl;

    else
        std::cout << "n mod 5 is not odd and is not 0" << std::endl;

    return 0;
}
```

Control sequences

Switch / case

Example:

```cpp
int main () {
    int n;
    std::cin >> n;

    switch (n % 5) {
        case 0:
            std::cout << "n div by 5" << std::endl;
            break;
        case 1:
        case 3:
            std::cout << "n mod 5 is odd" << std::endl;
            break;
        default:
            std::cout << "n mod 5 is even" << std::endl;
            break;
    }

    return 0;
}
```

Void

If you want to create a function, which will not return anything, you can use void as a return type:

```cpp
void say_hello () {
    std::cout << "Hello world!" << std::endl;
}

int main () {
    say_hello();
    return 0;
}
```

But still, you can return return keyword:

```cpp
void say_hello (int x) {
    if (x == 0)
        return;

    std::cout << "Hello world!" << std::endl;
}

int main () {
    say_hello(123);
    say_hello(0);
    return 0;
}
```

ASCII table

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[ENG OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*

ASCII table

Prints 97:

```cpp
int main () {
    std::cout << (int)'a' << std::endl;
    return 0;
}
```

Arithmetics:

```cpp
int main () {
    std::cout << 'a' + 'b' << std::endl;
    std::cout << 'b' - 'a' << std::endl;
    std::cout << 'a' + 5 << std::endl;
    std::cout << 'a' - 5 << std::endl;
    std::cout << (char)100 << std::endl;
    return 0;
}
```

```
195
1
102
92
d
```

Qualifiers

const qualifier:

Variables declared with const-qualified types are **not** modifiable. In particular, they are not assignable:

```
int main () {
    /// Not an assignment:
    const int x = 123;
    x -= 123;
    return 0;
}
```

```
void f (const int x) {
    x ++;
}
```

RULE

If you are not going to change the value of a variable, add const-qualifier to it. This will help you to avoid mistakes, because if you try to change the value, the code will not compile.

Types of errors in C++

Compilation error is a situation in which the compiler cannot convert your code to executable.

- Lexical error - the written code cannot be recognized by the compiler
- Syntax error. Natural language example: a sentence written from random words.
- Semantic error: The written text is correct, but it makes no sense. example from natural language: "Eat, please, that table." You cannot make sense and cannot complete the task given to you with the given objects.

Runtime errors can't be predicted at compile time:

- Access to unallocated memory.
- Stack overflow. stack memory overflow (will discuss later, but a simple example is infinite recursion)

Undefined behaviour (UB):

- This is when we write something that the C ++ language standard does not say anything about.

CE

1. Lexical: an error in the process of splitting into tokens, i.e. compiler saw after- a sequence of characters that could not be deciphered.
True (std) (::) (cout) (< <) (x) (;) is an example of correct splitting into tokens
False 24abracadabra;
2. Syntactic: Occurs when you write a statement that is invalid according to the grammar of the language (for example, the speech of Master Yoda)
False int const x = 5;
False x+5+;
False no semicolon (;)
False mismatch parentheses or curly braces
3. Semantic: occurs when the instruction is written correctly, but its compiler cannot perform (for example: eat yourself at this table)
False use of undeclared variables
False calling the size() method from a variable of type int
False x++ = a + b;
False call foo(3); although the signature is: void foo(int a, int b){}

Practice

Problem 1:
Input a number and output it’s bits in a sequence.

Problem 2:
Let’s consider sets of:

- Digits
- Latin letters (both capital and lowercase ones)
- +
- -

Implement the next functions:

- Input the set
- Output set
- Unite two sets
- Intersect two sets
- Invert set
- Calculate symmetric difference between two sets
- Calculate difference between two sets