

Astana IT University

Programming Fundamentals C++

Course characteristics

Discipline type: Basic discipline

Course duration: 1 trimester

Hours: 60 hours

Course requirements and materials:

Materials and requirements for this course

- a. PC or Laptop
- b. C++ Library (MinGW)
- c. Visual Studio IDE (optional)
- d. Sublime Text Editor or Notepad++ (optional)

Course description

This course is developed to learn basics of programming fundamentals and writing algorithms in C++ programming language. During this course, you will improve your programming skills, writing simple algorithms with using C++ technologies.

Course Plan

1. Basic syntax, variables, primitive data-types, operators, first C++ program
 - a. First program
 - b. Variables
 - c. Comments
 - d. Primitive data-types
 - e. Operators
 - f. Reading data from console
2. Decision making (if-else)
 - a. Condition if-else
 - b. else if
 - c. switch case
3. Loops
 - a. while loop
 - b. do-while loop
 - c. for loop
 - d. break/continue
4. Arrays. One dimensional arrays
 - a. One dimensional array
5. Multi-dimensional arrays
 - a. Two-dimensional arrays

6. Strings

- a. String class
- b. Strings as a char array

7. Functions and arguments, header files

- a. Functions (return type and void)
- b. Arguments
- c. Function prototype
- d. Function overloading
- e. Header files

8. Pointers & references

- a. Pointers
- b. References

9. Pointers as arrays (Dynamic arrays)

- a. Dynamic arrays
- b. c styled strings

Chapter 1

Basic syntax, variables, primitive data-types, operators, first C++ program

What is C++?

C++ is a free-form programming language that supports object-oriented, procedural and generic programming. C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. C++ was developed by Danish computer scientist **Bjarne Stroustrup** at Bell Labs since 1979 as an extension of the C language.

First program

Let's create a simple C++ program, that asks user to input 2 numbers and outputs sum of them:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int a,b;
7     cout<<"Insert a: "<<endl;
8     cin>>a;
9     cout<<"Insert b: "<<endl;
10    cin>>b;
11
12    cout<<a+b<<endl;
13
14    return 0;
15 }
```

After execution, if you input two numbers, for example 4 and 5, the program will output 9 as a result.

Normally, the program starts with including libraries and using namespaces, such as **std**. Namespace **std** is needed to use **cin>>** and **cout<<** commands for inputting and outputting data respectively. To execute the program we need to create a **main()** function. Inside of this function, we will write our first program. The command **cin>>** is used to insert numbers from console. After inserting number, you will output their sum by using the command - **cout<<**.

Variables

A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times. Normally, in classic math you create a variable to store some numbers and reuse it again later. Here an example of creating variable:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int a = 10;
7     int b = 20;
8
9     int sum = a+b;
10
11     cout<<sum<<endl;
12
13     return 0;
14 }
```

In this example, we created two integer variables, a and b. Then, we create a new variable, called sum, which equals to a+b. In the end, we output them, by using command **cout<<**.

So, when we create variables a, b and sum, we instantiate them at memory locations, and reuse this locations again.

Comments

The C++ comments are statements that are not executed by the compiler. The comments in C++ programming can be used to provide explanation of the code, variable, method or class.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4
5     double x = 2.4; // Create double variable x
6     // Output the square of x
7     cout<<x*x<<endl;
8
9     return 0;
10 }
```

You may write comments anywhere, by typing `//` (double slash). This kind of comments we call single line comment.

If you want to write a long comments with multiple lines, you will use `/* */` commands.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     /*
7         This program is created
8         to develop a program
9         that multiplies variable x
10        to itself
11    */
12
13    double x = 2.4;
14
15    cout<<x*x<<endl;
16
17    return 0;
18 }
```

Primitive data-types

The primitive data-type specifies the size and type of information the variable will store. The primitive data-types are integer-based and floating-point based.

Type	Keyword	Typical Bit Width	Typical Range
Boolean	bool	1 byte	true and false
Character	char	1 byte	0 to 255
Integer	int	4 bytes	-2147483648 to 2147483647
Float	float	4 bytes	+/- 3.4e +/- 38 (~7 digits)
Double	double	8 bytes	+/- 1.7e +/- 308 (~15 digits)

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int a = 10;
7     double b = 3.45;
8     float c = 22.4;
9     char d = 'A';
10    bool e = true;
11
12    cout<<a<<endl;
13    cout<<b<<endl;
14    cout<<c<<endl;
15    cout<<d<<endl;
16    cout<<e<<endl;
17
18    return 0;
19 }
```

Strings

If you want to use text variables, you may use string elements. A string variable contains a collection of characters surrounded by double quotes.

Example:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     string university = "AITU";
7     cout<<"Hello " <<university<<endl;
8
9     return 0;
10 }
```

You may add two strings into one. This process is called **concatenation**. The + operator can be used between strings to add them together to make a new string.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     string name = "Ilyas";
7     string surname = "Zhuanyshev";
8
9     string fullName = name + " " + surname;
10
11     cout<<"You are "<<fullName<<endl;
12
13     return 0;
14 }
```

Be careful on declaring data-types. You must follow the rules of creating variables according to their types. You cannot create a character variable with a text value or declare integer as a text.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     char x = 'Astana IT University';
7     int y = "Nur-Sultan";
8
9     return 0;
10 }
```

Such kind of declaration is incorrect, and compiler will return you an error.

Operators

To perform operations, we use operators. In C++ we have several types of operators, such as arithmetic, logical, or assignment.

Here, we have a list of operator types:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators

Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations, such as adding, subtracting or dividing. Assume that we created two variables, **a = 60** and **b = 30**:

Operator	Description	Example
+	Adds two operands	a + b will give 90
-	Subtracts second operand from the first	a - b will give 30
*	Multiplies both operands	a * b will give 1800
/	Divides numerator by denominator	a / b will give 2
%	Modulus Operator and remainder of after an integer division	a % b will give 0
++	Increment operator, increases integer value by one	a++ will give 61
--	Decrement operator, decreases integer value by one	a-- will give 59

Here an example of using arithmetic operations in code:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int a = 200;
7     int b = 33;
8
9     cout<<a+b<<endl; // result will be 233
10    cout<<a-b<<endl; // result will be 167
11    cout<<a*b<<endl; // result will be 6600
12    cout<<a/b<<endl; // result will be 6
13    cout<<a%b<<endl; // result will be 2
14    a++;
15    cout<<a<<endl; // result will be 201
16    b--;
17    cout<<b<<endl; // result will be 32
18
19    return 0;
20 }
```

Relational Operators

Relational operators are used to compare variables, to get as a result boolean value, true or false. Assume that we created two variables, **a = 60** and **b = 30**:

Operator	Description	Example
==	If the values of two operands are equal or not	a==b will give false
!=	If the values of two operands are not equal or equal	a!=b will give true
>	If the left operand is greater than the right operand	a > b will give true
<	If the left operand is less than the right operand	a < b will give false
>=	If the left operand is greater or equal to the right operand	a >= b will give false
<=	If the left operand is less or equal to the right operand	a++ will give 61

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int a = 200;
7     int b = 33;
8
9     cout<<(a==b)<<endl; // result will be 0 (false)
10    cout<<(a!=b)<<endl; // result will be 1 (true)
11    cout<<(a>b)<<endl; // result will be 1 (true)
12    cout<<(a<b)<<endl; // result will be 0 (false)
13    cout<<(a>=b)<<endl; // result will be 1 (true)
14    cout<<(a<=b)<<endl; // result will be 0 (false)
15
16    return 0;
17 }
```

Logical Operators

Logical operators are used to determine the logic between variables or values. Assume that we created two boolean variables, **a = true** and **b = false**:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are true, then condition becomes true.	a&&b will give false
 	Called Logical OR Operator. If any of the two operands is true, then condition becomes true.	a b will give true
!	Called Logical NOT Operator. Use to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!b will give true

We can understand it logically, from following statements:

I am telling: "Me **and** Dimash are both programmers". According to this statement, if one of us is not a programmer, my statements will be wrong (false). But, if we all are programmers, my statement will be correct (true).

Let's look at the another statement. I am telling: "Me **or** Dimash is a programmer". According to this statement, if one of us is a programmer, my statement will be correct (true). But, if we all are not programmers, my statement will be incorrect (false).

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows:

p	q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume that we created two variables, **a = 60**, and **B = 13**. Now in binary format they will be as follows:

a = 0011 1100

b = 0000 1101

a&b = 0000 1100, a|b = 0011 1101, a^b = 0011 0001

Operator	Description	Example
&	Binary AND operator	a&b will give 12 (0000 1100)
	Binary OR Operator	a b will give 61 (0011 1101)
^	Binary XOR Operator	a^b will give true (0011 0001)

Assignment Operators

Assignment operators are used to assign values to variables. Assume that we created two variables, **a = 60**:

Operator	Description	Example
=	Simple assignment operator	a = 14; a will be 14
+=	Add and assignment operator	a+=2 will be a = a + 2 => 62
-=	Subtract and assignment operator	a-=2 will be a = a - 2 => 58
=	Multiply and assignment operator	a=2 will be a = a * 2 => 120
/=	Divide and assignment operator	a/=2 will be a = a / 2 => 30
%=	Modulus and assignment operator	a%=7 will be a = a%7 => 4

You may also increment and decrement character elements. Characters can be represented as integers according to their ASCII code.

Here your can see ASCII table below:

0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	\$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(72	H	104	h
9	TAB (horizontal tab)	41)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[123	{
28	FS (file separator)	60	<	92	\	124	
29	GS (group separator)	61	=	93]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

According to this table, decimal value of a character 'A' is 65, 'B' - 66. If we create a character element with value, for example 'a', and increment it by one, the value will be 'b'.

main.cpp

```
1 #include <iostream>
2
3 using namespace std;
4 int main() {
5
6     char x = 'a';
7     x++;
8     cout<<x<<endl;
9     x+=10;
10    cout<<x<<endl;
11
12    return 0;
13 }
```

The output of the following code will be:

```
b
1
```

Reading data from console

You may insert variables from console, by using **std::cin>>** command (If you are using namespace std, the you may just write **cin>>**).

main.cpp

```
1 #include <iostream>
2
3 using namespace std;
4 int main() {
5
6     int a,b;
7     cin>>a>>b;
8     cout<<a+b<<endl;
9
10    return 0;
11 }
```

After execution, console will wait when you insert your numbers a and b, then executes it.

std::cin>> command can read any data type, it doesn't matter, double, integer or string.

main.cpp

```
1 #include <iostream>
2
3 using namespace std;
4 int main() {
5
6     int a,b;
7     string name;
8     double c;
9     cin>>a>>b>>name>>c;
10
11    cout<<name<<": "<<(a+b)<<" "<<c<<endl;
12
13    return 0;
14 }
```

If you type in console:

```
4 5 Ilyas 2.3
```

Your output will be:

```
Ilyas 9 2.3
```

using getline() function

There may be situation you want to type as a text multiple words. For example, **My name is Ilyas, and I am from Kazakhstan**. If you use `std::cin>>` command, you may take only first word.

main.cpp

```
1 #include <iostream>
2
3 using namespace std;
4 int main() {
5
6     string text;
7     cin>>text;
8     cout<<"My sentence is: "<<text<<endl;
9
10    return 0;
11 }
```

If you write in console:

My name is Ilyas, and I am from Kazakhstan

Your output will be:

My sentence is: My

The real problem is that command `std::cin>>` considers a space (whitespace, tabs, etc) as a terminating characters. To read whole line you can use special function, called – **getline()**

main.cpp

```
1 #include <iostream>
2
3 using namespace std;
4 int main() {
5
6     string text;
7     getline(cin, text);
8     cout<<"My sentence is: "<<text<<endl;
9
10    return 0;
11 }
```

If you write in console:

Ernis and Abay played football against Ilyas and Yerzhan

Your output will be:

My sentence is: Ernis and Abay played football against Ilyas and Yerzhan

We often use the **getline()** function to read a line of text. It takes **cin** as the first parameter, and the **text** variable as second.

Practice works

Basic syntax, variables, primitive data-types, operators, first C++ program

Practice task 1

Write a program that prompts the user to input numbers **x** and **y**. Then calculates the difference of values of inputted two numbers and prints the result.

Input:

2.2

7.5

Output:

-5.3

Practice task 2

Write a program that prompts the user to input a value of the variable **x**. Then the program should calculate and print the value of **y** according to the function: $y = 4x^2 - 4x + 2$

Input:

5

Output:

82

Practice task 3

Write a program that prompts the user to input a value of the variable **L** in meters. Then the program should convert this **L** into miles. 1 mile = 1609.34 meters.

Input:

100

Output:

0.062137 miles

Input:

5000

Output:

3.10686 miles

Practice task 4

Write a program that prompts the user to input salaries of Ilyas, Dimash and Askar in KZT. Then the program should find the average salary of them.

Input:

300000

95000

400000

Output:

265000 KZT

Practice task 5

Write a program that inputs a five-digit integer, separates the integer into its individual digits and prints the digits separated from one another by three spaces each.

Input:

12345

Output:

1 2 3 4 5

Input:

76548

Output:

7 6 5 4 8

Practice task 6

Write a program that prompts the user to input a value of the variable x. Then the program should calculate and print the value of y according to the function: $y = 4x^3 - 3x^2 + 2$

Input:

3

Output:

92

Practice task 7

Write a program that prompts the user to input name, surname, age and country. Then program should output user data according to following format:

Input:

Ilyas

Zhuanyshev

30

Kazakhstan

Output:

You are Ilyas Zhuanyshev, 30 years old from Kazakhstan

Practice task 8

Write a program that prompts the user to input length of sides of right triangle - a and b. Program should output the length of hypotenuses. (**Hint: to get square root, you will use the special function, called **sqrt()**, from **<cmath>** library. Ex: **sqrt(9)** will return 3)

Input:

9 12

Output:

5

Chapter 2

Decision making (if-else)

Condition if-else

Decision making is used to specify one or more conditions to be evaluated or executed by program. In C++ programming language decision making is realized by **if** condition. **if** condition is used to test the condition. It checks boolean condition: true or false.

The structure of code:

```
if(condition){  
    //code to be executed  
}
```

Example:

main.cpp

```
1 #include <iostream>  
2 using namespace std;  
3  
4 int main() {  
5  
6     int a = 50;  
7  
8     if(a>40) {  
9         cout<<"a is greater than 40"<<endl;  
10    }  
11  
12    cout<<"Program ended"<<endl;  
13  
14    return 0;  
15 }
```

In this program, we created an integer variable `a = 50`. On line 8 we check **if this variable a is greater than 40 or not**. If condition is true, then our program will output a text **"a is greater than 40"**, otherwise program will skip this line.

The output will be:

```
a is greater than 40  
Program ended
```

Try to change the value of variable a = 50 into 30, then test it.

main.cpp

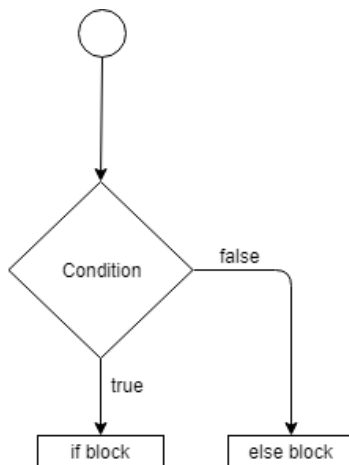
```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int a = 10;
7
8     if(a>40){
9         cout<<"a is greater than 40"<<endl;
10    }
11
12    cout<<"Program ended"<<endl;
13
14    return 0;
15 }
```

The output will be:

Program ended

The **"else block"** is used to perform operations if our condition is false. You will just write else after **"if block"** and open curly braces.

The scheme of if-else conditions is looked like this:



Example:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int a = 10;
7
8     if(a>40){
9         cout<<"a is greater than 40"<<endl;
10    }else{
11        cout<<"a is not greater than 40"<<endl;
12    }
13
14    cout<<"Program ended"<<endl;
15
16    return 0;
17 }
```

In this example, our output will be:

```
a is not greater than 40
```

```
Program ended
```

if else if

An "if condition" can be followed by an optional "else if else" condition, which is very useful to test various conditions. Imagine that, you are going to hire a programmer to your company. You have one vacancy and four candidates to this position. You ask your first candidate for some questions about his hard and soft skills. If, this candidate does not fit to your company, you will search and ask for questions to the next candidate. If next candidate also does not fit to your company, you ask third your candidate for questions. However, if your third candidate is really fits to your company, you will hire him. You will not ask questions to the fourth candidate, because you have already found your programmer. Same logic in "else if else" condition.

Let us look at the example about marks and grade. I will input my grade, taken from exam, and my program will output the mark in letters. The graduation policy will be like this:

90 or higher - A, 80 or higher - B, 70 or higher - C, 50 or higher D otherwise F.

Example:

main.cpp

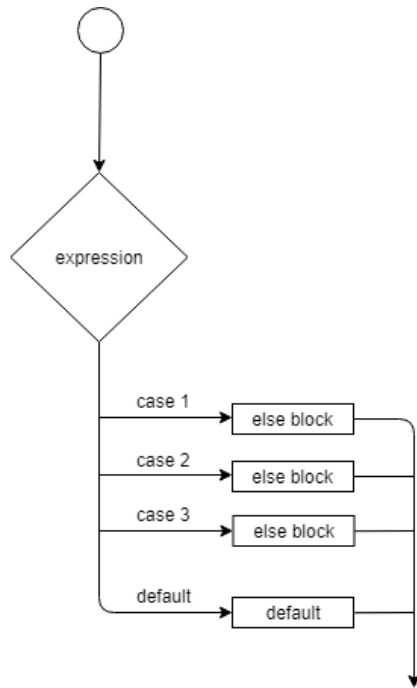
```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int grade;
7     cin>>grade;
8
9     if(grade>=90) {
10
11         cout<<"You got A from exam"<<endl;
12
13     }else if(grade>=80) {
14
15         cout<<"You got B from exam"<<endl;
16
17     }else if(grade>=70) {
18
19         cout<<"You got C from exam"<<endl;
20
21     }else if(grade>=50) {
22
23         cout<<"You got D from exam"<<endl;
24
25     }else{
26
27         cout<<"You failed, F"<<endl;
28
29     }
30
31     return 0;
32 }
```

So, in execution, if you input 77, the output will be C.

switch case

A switch statement allows a variable to be tested for equality against a list of values. I can use it in such tasks, when I really have concrete answer values and variants.

Here, we can see flow diagram of "**switch case statement**" below:



Let's look at the example about ordering drink below:

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int order;
7     cout<<"What would you like to drink?"<<endl;
8     cout<<"1 - Coffee, 2 - Tea, 3 - Juice, otherwise Water"<<endl;
9     cin>>order;
10
11     switch(order) {
12         case 1:
13             cout<<"You ordered coffee"<<endl;
14             break;
15         case 2:
16             cout<<"You ordered tea"<<endl;
17             break;
18         case 3:
19             cout<<"You ordered juice"<<endl;
20             break;
21         default:
22             cout<<"You will drink water"<<endl;
23     }
24
25     return 0;
26 }
```

Practice works

Decision making (if-else)

Practice task 1

Write a program that prompts the user to input numbers a and b. The program should output which number is greater and for how much more.

Input:

5 6

Output:

6

Input:

7 22

Output:

22

Practice task 2

Write a program that prompts the user to input a value of the resistors R1 and R2. Then the program should calculate and print the total resistance RT. Resistors can be located in either way: 1 – series, 2 – parallel. (**Hint: remember the formula of total resistance parallel resistors and series resistors)

Input:

10 15 2

Output:

6

Input:

10 15 1

Output:

25

Practice task 3

Write a program that prompts the user to input a total price of the goods in supermarket. Then the program should print the receipt taking into account the fact that if the total price is greater than 500 USD, the user deserves 10% discount, otherwise, user will pay full price.

Input:

120

Output:

TOTAL: 120 USD

Thank you!

Input:

520

Output:

TOTAL: 468 USD

Thank you!

Practice task 4

Write a program that prompts the user to input two numbers. Then the program should check whether the first number is greater, less or they are equal.

Input:

10 2

Output:

10 is greater than 2

Input:

2 20

Output:

2 is less than 20

Input:

10 10

Output:

They are equal

Practice task 5

A tourist walked **D** km for a day. Before noon, he walked **t1** hours and 15 km. After noon, he walked **t2** hours.

When the velocity of the tourist was greater, before or after noon? Write a program that prompts the user to input the distance **D**, the amount of time **t1**, **t2** and output the answer to the above question.

Input:

21 5 3

Output:

Before

Input:

40 5 7

Output:

After

Practice task 6

Write a program that prompts the user to input x and y. Then the program should check will the graph of the function: $y = 2x^2 - 3x + 2$ will pass through the inputted coordinates.

Input:

2 4

Output:

Yes

Input:

4 7

Output:

No

Practice task 7

Write a program that prompts the user to input areas of the circle and square. The program should check whether the square can be placed inside of the circle.

Input:

43 33

Output:

No

Input:

20 10

Output:

Yes

Practice task 8

Write a program that prompts the user to input amount of money in KZT and number of currency. The program should output the exchange value of inputted KZT amount into chosen currency.

1 – USD, 2 – EUR, 3 – GBP (1 USD = 380 KZT, 1 EUR = 420 KZT, 1 GBP = 480 KZT)

Input:

2000 1

Output:

5.26 USD

Input:

4000 3

Output:

8.33 GBP

Practice task 9

Write a program that prompts the user to input two numbers, a and b. The program should say if a is divisible by b or not. (**Hint: Use modulus (%) operator, to find remainder)

Input:

5 3

Output:

Not divisible

Input:

100 5

Output:

Divisible

Practice task 10

Write a simple program of authentication, where user will input own login and password. If login is "ilyas" and password is "qwerty" then you will be authenticated, otherwise no.

Input:

ilyas

realmadrid

Output:

Incorrect login or password

Input:

ilyas

qwerty

Output:

Welcome ilyas, you are authenticated!

Practice task 11

Write a program that determines whether a number is lucky or not if sum of given six digit number's first three digits equals to the sum of the last three digits of that number. It is guaranteed that we will type only six-digit number.

Input:

123456

Output:

No

Input:

434065

Output:

Yes

Chapter 3

Loops

while loop

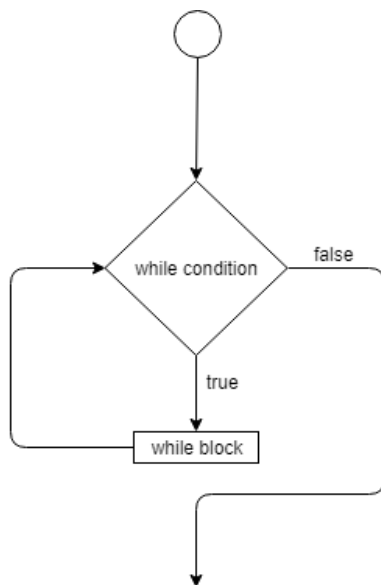
There may be a situation, when you need to execute a block of code several times. Imagine that, in real life, you have an aim - to become super athlete or runner. If you want to become a super athlete, you have to train a lot systematically. Every day you do your exercises and repeat it at least for 1 year. So, here 1 day of exercise is a 1 iteration, and your exercise is your operation of your loop. You will do this exercise 365 times (1 year = 365 days). The same logic in loops. You will repeat your operation until when you reach the end of your loop.

The code structure:

```
while(condition){  
    //operation  
}
```

If condition is true, we will execute an operation inside of "**while block**" and return to condition, until this condition will not become false.

The flow diagram of while block:



Now, let's look at the example, where I will print 10 times "Hello Astana IT University":

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int i = 0;
7
8     while(i<10) {
9         cout<<"Hello Astana IT University"<<endl;
10        i++;
11    }
12
13    return 0;
14 }
```

The output will be:

```
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
```

In this example we created an integer **i = 0**, which we compare with a number 10. It means, while our integer **i** is less than 10, we will print - "Hello Astana IT University". Inside of our **"while block"**, after printing a text, we increase a value of integer **i** by 1. So, after one iteration, our **i** will become 1. After next iteration, **i** will be 2, and then 3, 4, etc. Our loop will stop when **i** become 10, because **10<10** is false. When condition become false, we will skip our loop.

Let's try to look at another example, where we will print all odd numbers between 100 and 1000. It means, all numbers with sequence 101, 103, 105, ..., 995, 997, 999. We see that our loop must start from 101, and after every iteration, we will increase our number by 2, until this number to be less than 1000.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int i = 101;
7
8     while(i<1000) {
9         cout<<i<<" ";
10        i=i+2; // Or you may write i+=2;
11    }
12
13    return 0;
14 }
```

do-while loop

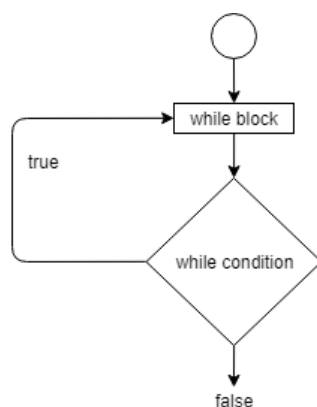
The do-while loop is a variant of the while loop. This loop will execute the code block once before checking if the condition is true, then it will repeat the loop as long as the condition is true.

The code structure:

```
do {
    // operation
}
while (condition);
```

Firstly, you do your operation, and if your condition is true, you will repeat it again until your condition to be true. Notice that if our condition is false in the beginning, we will do our operation at least once.

Flow diagram of do-while:



Let's see an example

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int i = 0;
7
8     do{
9         cout<<"Hello Astana IT University"<<endl;
10        i++;
11    }while(i<10);
12
13    return 0;
14 }
```

The output will be:

```
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
Hello Astana IT University
```

Do not confuse with a number operations. Here we also operate 10 times (print our text) with a difference when our condition is false in the beginning, we will print our text at least once.

for loop

For loop is a simplified version of our previous while loop with different syntax.

The code structure:

```
for(init; condition; increment){
    // operation
}
```

We have 3 parts of for loop:

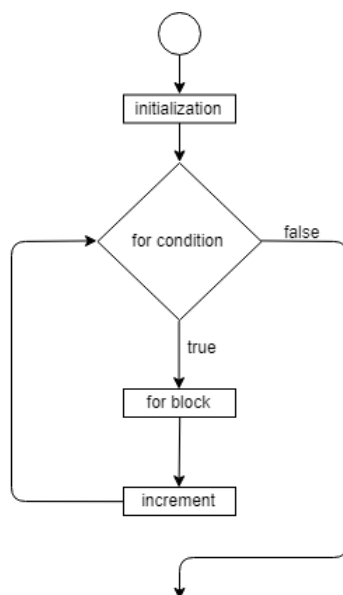
1. init part
2. condition part
3. increment part

The **init part** is executed first, and only once. This step allows you to declare and initialize any loop control variables.

Condition part will evaluate after **init part**. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute.

After the body of the for loop executes, the flow of control jumps back up to the **increment part**. This part can be left blank, as long as a semicolon appears after the condition.

Flow diagram:



Let's see an example:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     for(int i=0;i<10;i++){
7         cout<<"Hello Astana IT University"<<endl;
8     }
9
10    return 0;
11 }
```

break/continue

The **break** statement can be used to jump out of a loop.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     for(int i=0;i<10;i++) {
7         if (i==5) {
8             break;
9         }
10        cout<<i<<endl;
11    }
12    return 0;
13 }
```

The output will be:

```
0
1
2
3
4
```

As you see from example, our loop does not iterate 10 times, because when an integer *i* becomes 5, we call **break** command, to jump out of the loop.

The **continue** statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop. It is similar to skipping one iteration.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     for(int i=0;i<10;i++) {
7         if (i==5) {
8             continue;
9         }
10        cout<<i<<endl;
11    }
12    return 0;
13 }
```

The output will be:

```
0
1
2
3
4
6
7
8
9
```

As you see from example, when `i` becomes 5, we call **`continue`** command to skip this iteration, and return back to the loop.

In future, you will often use loops to create own algorithms or complete several goals in application development. Loops are used in all popular programming languages, such as C++, C#, Java, Python, Ruby, PHP, JavaScript, Kotlin and etc., because it is one of the most fundamental parts of programming language construction.

Practice works

Loops

Practice task 1

Write a program that asks a user to input integer n. Program should output all even elements, starting from 0 to n included.

Input:

9

Output:

0 2 4 6 8

Input:

22

Output:

0 2 4 6 8 10 12 14 16 18 20 22

Practice task 2

Write a program that prompts the user to input an integer n. Program should output all integers from n to 0 in reverse order.

Input:

9

Output:

9 8 7 6 5 4 3 2 1 0

Input:

16

Output:

16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Practice task 3

Write a program that asks a user to input an integer n . The program should output product of the first n numbers.

Input:

5

Output:

120

Input:

10

Output:

3628800

Practice task 4

Write a program that prompts the user to an integer n . Then the program should output the sum of the first n elements of the sequence: $1 + 1/2 + 1/3 + 1/4 + \dots + 1/n$

Input:

3

Output:

1.83333

Input:

10

Output:

2.92897

Practice task 5

Write a program that prompts the user to an integer n . Then the program should output the sum of the first n elements of the sequence: $1 - 1/3 + 1/5 - 1/7 + 1/9 - \dots$

Input:

3

Output:

1.86667

Input:

100

Output:

0.782898

Practice task 6

Write a program that prompts the user to an integer n . Then the program should output the sum of the first n elements of the sequence: $2 - 4 + 6 - 8 + 10 - 12 \dots$

Input:

3

Output:

4

Input:

15

Output:

16

Practice task 7

Write a program that asks a user to input integers. The program stops reading integers when the user inputs 0. The program should output sum of the inputted integers and the average.

Input:

1 2 3 4 5 0

Output:

15

3

Input:

10 44 53 27 0

Output:

134

33.5

Practice task 8

Write a program that asks a user to input integers. The program stops reading integers when the user inputs 0. The program should output sum of the inputted integers and the average.

Input:

1 2 3 4 5 0

Output:

15

3

Input:

10 44 53 27 0

Output:

134

33.5

Practice task 9

Write a program that prompts the user to integers. The program stops reading integers when the user inputs 0. The program should output the maximum and minimum element among inputted numbers

Input:

10 22 1 -5 16 0

Output:

22

-5

Input:

9 9 9 9 9 9 0

Output:

9

9

Practice task 10

Write a program that prompts the user to an integer n. Then the program should output the sum of the first n elements of the sequence 1, 3, 5, 7 . . .

Input:

10

Output:

100

Input:

15

Output:

225

Practice task 11

Write a program that asks a user to input numbers. The program should stop reading numbers when the user inputs 0. The program should output the product of the inputted numbers.

Input:

2.2 5.6 7.8 0

Output:

96.096

Input:

3 4 5 6 7 0

Output:

2520

Practice task 12

Write a program that asks a user to input integers. The program should stop reading numbers when the user inputs 0. The program should output the sum of the odd elements among inputted integers.

Input:

1 2 3 4 5 6 7 8 9 0

Output:

25

Input:

-7 5 8 2 9 0

Output:

7

Practice task 13

Write a program that prompts the user to an integers a. Then the program should output the sum of digits of the inputted integer.

Input:

726

Output:

15

Input:

229

Output:

13

Practice task 14

Write a program that prompts the user to an integers a and b. Then the program should output the result of a*b (multiplication of a and b) without using * operator.

Input:

4 5

Output:

20

Practice task 15

Write a program that prompts the user to an integer a in binary format. Then the program should convert inputted number into decimal.

Input:

1000

Output:

8

Input:

11011

Output:

27

Chapter 4

Arrays. One dimensional arrays

One dimensional array

Consider the case where you want to store the exam scores for 5 students in a group and print their total and average value. Normally, you may create 5 double variables in your program and operate with them.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     double sc1 = 100;
7     double sc2 = 90;
8     double sc3 = 95;
9     double sc4 = 80;
10    double sc5 = 75;
11
12    double total = sc1+sc2+sc3+sc4+sc5;
13    double average = total/5;
14
15    cout<<"Total score : "<<total<<endl;
16    cout<<"Average score : "<<average<<endl;
17
18    return 0;
19 }
```

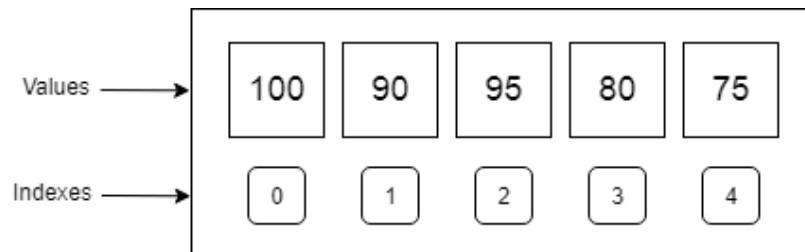
As you see from example, we created double variables for each score of student. However, unfortunately, it is ineffective, because creating 5 variables for each student to store their exam score is too expensive. In addition, if you want to store scores of 100 students, you have to create 100 double variables. It is too expensive.

One of the solutions of this problem is - arrays. You may create one dimensional double array with a size 5, and store all scores in one variable.

An array is a **series of elements of the same type placed in contiguous memory locations** that can be individually referenced by adding an index to a unique identifier. Normally, arrays are used in all popular programming languages, but code implementation may be a little bit

different. C++ provides an array, which stores a fixed-size sequential collection of elements of the same type.

In real life, you may get an example of one-man bicycle and bus. One-man bicycle is a normal data-type variable (int, double, float etc.), which can seat one man, but a bus is an array of data-type variable, which can seat many people. The same logic in arrays, they can hold a sequence of data-types, according to given size in one array variable.



In C++, array index starts from 0. We can store only fixed set of elements in C++ array.

To create an array of 5 doubles, you could write:

```
double scores[5] = {100, 90, 95, 80, 75};
```

Here an example of creating simple double array, with a size 5, where we store scores of students from exam:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     double scores[5] = {100, 90, 95, 80, 75};
7
8     double total = 0;
9
10    for(int i=0;i<5;i++){
11        total+=scores[i];
12    }
13
14    double average = total/5;
15
16    cout<<"Total score : "<<total<<endl;
17    cout<<"Average score : "<<average<<endl;
18
19    return 0;
20 }
```

Let's try to understand, what we did in this code. Firstly, we created a double array, with a size 5 elements, by typing a code **double students[5] = {100, 90, 95, 80, 75}**. To access for the first element, you will write **students[0]**, because index starts from 0. The second element – **students[1]** and etc. If you want to print the last element of array, you will write **cout<<students[4]<<endl**; According to this logic, we create a loop, starting from 0 to 5 not included. It means, our integer **i** will be 0, 1, 2, 3, 4 at each iteration. So, we can access to all element of array, and get their sum, by collecting values into variable **total = 0**. In the end, dividing total by size of array (5), we can get an average value. This approach is better than creating 5 double variables for each score. In future, you may manipulate with your array, by creating own algorithms.

There are several advantages of using arrays:

1. *Code Optimization (less code)*
2. *Random Access*
3. *Easy to traverse data*
4. *Easy to manipulate data*
5. *Easy to sort data etc.*

You may declare an array, without putting elements directly on it:

```
double scores[5];
```

You will put elements later, by accessing to each index:

```
double scores[5];  
  
scores[0] = 100;  
scores[1] = 95;  
scores[2] = 90;  
scores[3] = 80;  
scores[4] = 75;
```

You may declare a size of array and insert elements from console, manually.

Let's look at the example, given below:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int size;
7     cin>>n;
8
9     double scores[n];
10
11     for(int i=0;i<n;i++){
12         cin>>scores[i];
13     }
14
15     double total = 0;
16
17     for(int i=0;i<n;i++){
18         total+=scores[i];
19     }
20
21     double average = total/n;
22
23     cout<<"Total score : "<<total<<endl;
24     cout<<"Average score : "<<average<<endl;
25
26     return 0;
27 }
```

We create an integer **n**, and insert its value from console. Then, we declare an array with size **n**. By opening loop, we insert each element of array consequently from console. In the end, we calculate average and total value as we did before.

You may traverse the array by using foreach loop:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4
5     double scores[5] = {100, 90, 95, 80, 75};
6
7     for(double x: scores){
8         cout<<x<<endl;
9     }
10    return 0;
11 }
```

Here we create local variable **x**, which is an element of array in each iteration.

In future topics, you will see multi-dimensional arrays, but in this chapter, you will work only with one-dimensional array.

Practice works

Arrays. One dimensional arrays

Practice task 1

Write a program that asks a user to input an integer n. The program should prompt the user to input n integers and store them in one-dimensional array. Then the program should output elements in reverse order.

Input:

10
1 2 3 4 5 6 7 8 9 10

Output:

10 9 8 7 6 5 4 3 2 1

Practice task 2

Write a program that asks a user to input an integer n. The program should prompt the user to input n integers and store them in one-dimensional array. Then the program should output the sum and average of the elements in the array.

Input:

10
1 2 3 4 5 6 7 8 9 10

Output:

55
5.5

Practice task 3

Write a program that asks a user to input an integer n. The program should prompt the user to input n integers and store them in one-dimensional array. Then the program should output the minimum and maximum elements in the array.

Input:

10
24 2 32 4 -5 61 17 28 9 12

Output:

-5
61

Practice task 4

Write a program that asks a user to input an integer n. The program should prompt the user to input n integers and store them in one-dimensional array. Then the program should output only positive integers in the array.

Input:

10

2 -4 6 -9 24 -5 74 34 -23 45

Output:

2 6 24 74 34 45

Practice task 5

Write a program that asks a user to input an integer n. The program should prompt the user to input n integers and store them in one-dimensional array. Then the program should swap the minimum with maximum and output the array.

Input:

10

2 -4 6 -9 24 -5 74 34 -23 45

Output:

2 -4 6 -9 24 -5 -23 34 74 45

Practice task 6

Write a program that asks a user to input an integer n. The program should prompt the user to input n integers and store them in one-dimensional array. Then the program should output the product of all non-zero elements in the array.

Input:

10

2 -4 6 -9 24 -5 74 34 -23 45

Output:

2877120

Practice task 7

Write a program that asks a user to input an integer n . The program should prompt the user to input n integers and store them in one-dimensional array. Then the program should output the geometric mean of all non-zero elements in the array. (**Hint: use **pow()** function)

Input:

10
6 19 0 -3 4 8 0 -6 9 5

Output:

6.43899

Practice task 8

Write a program that asks a user to input an integer n . The program should prompt the user to input n integers and store them in one-dimensional array. Further, the program should read an integer m and output all elements of the array that are greater than m .

Input:

10
2 -4 6 -9 24 -5 74 34 -23 45
2

Output:

6 24 74 34 45

Practice task 9

Write a program that asks a user to input an integer n . The program should prompt the user to input n integers and store them in one-dimensional array. Then the program should find the sum of elements between two zeros. It is guaranteed that array will include two zeros.

Input:

10
61 44 0 1 8 -5 6 0 9 5

Output:

10

Practice task 10

Write a program that asks a user to input an integer n. The program should prompt the user to input n integers and store them in one-dimensional array. The program should find the sum of elements that are located between maximum and minimum elements.

Input:

10

6 19 0 -3 4 8 0 -6 9 5

Output:

9

Practice task 11

Write a program that reads an integer n and n integers. All integers should be stored in an array. The program should output the elements that are located between first two negative integers. It is guaranteed that there will be at least 2 negative elements.

Input:

7

0 12 -5 4 20 -1 3

Output:

4 20

Practice task 12

Write a program that asks user to input integers. Program will stop asking, when user inputs 0. You must store all inserted elements in array. Print an amount of inserted numbers and roots of all elements in reverse order.

Input:

1 2 16.3 0

Output:

3

4.0373

1.4142

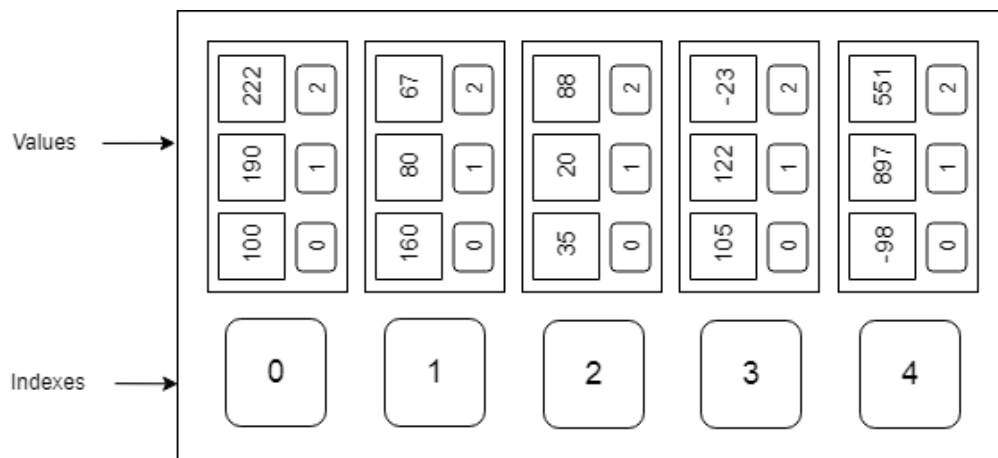
1

Chapter 5

Multi-dimensional arrays

Two-dimensional arrays

We know that the elements of an array can be any data type. Imagine that what will be if the **elements of an array will be array**. It looks like this:



In this picture, you see indexes from 0 to 4. Values of an array are arrays. It means we created two-dimensional array with size 5 and 3. The value at index [1][2] is 67. The value at [3][1] is 122 and etc. The code implementation will be like this.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4
5     int arr[5][3] = {
6         {100, 190, 222},
7         {160, 80, 67},
8         {35, 20, 88},
9         {105, 122, -23},
10        {-98, 897, 551}
11    };
12    cout<<arr[1][2]<<endl;
13    cout<<arr[3][1]<<endl;
14
15    return 0;
16 }
```

The output will be:

67
122

The multi-dimensional array is also known as rectangular arrays in C++. It can be two-dimensional or three-dimensional. The data is stored in tabular form (row * column) which is also known as matrix. An **array of arrays** is called a multidimensional array.

You may also take an analogy from a house with 5 floors. Each floor contains 3 flats. According to this example, the number **-23 lives in 3rd floor and 2nd flat**.

0th floor	100	190	222
1st floor	160	80	67
2nd floor	35	20	88
3rd floor	105	122	-23
4th floor	-98	897	551
	Flat 0	Flat 1	Flat 2

To print this number, you will write:

```
cout<<arr[3][2]<<endl;
```


Finally, you may represent this array as a matrix or a table:

	0	1	2
0	100	190	222
1	160	80	67
2	35	20	88
3	105	122	-23
4	-98	897	551

All possible indexes of my two-dimensional array are:

```
[0][0] [0][1] [0][2] //row 0
[1][0] [1][1] [1][2] //row 1
[2][0] [2][1] [2][2] //row 2
[3][0] [3][1] [3][2] //row 3
[4][0] [4][1] [4][2] //row 4
```

To print all numbers in array, you will use two-dimensional loops:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4
5     int arr[5][3] = {
6         {100, 190, 222},
7         {160, 80, 67},
8         {35, 20, 88},
9         {105, 122, -23},
10        {-98, 897, 551}
11    };
12    for(int i=0;i<5;i++){
13        for(int j=0;j<3;j++){
14            cout<<arr[i][j]<<" ";
15        }
16        cout<<endl;
17    }
18
19    return 0;
20 }
```

The output will be:

```
100 190 222
160 80 67
35 20 88
105 122 -23
-98 897 551
```

Like in one-dimensional array, it is not necessary to insert all elements on declaring two-dimensional array. You may declare your two-dimensional array and assign numbers later.

You will insert two numbers **n** and **m** from console as a size of your two-dimensional array. Then, by using loops, you will assign all values from console.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4
5     int arr[100][100];
6
7     int n,m;
8     cin>>n>>m;
9
10    for(int i=0;i<n;i++){
11        for(int j=0;j<m;j++){
12            cin>>arr[i][j];
13        }
14    }
15
16    for(int i=0;i<n;i++){
17        for(int j=0;j<m;j++){
18            cout<<arr[i][j]<<" ";
19        }
20        cout<<endl;
21    }
22
23    return 0;
24 }
```

Practice works

Multi-dimensional arrays

Practice task 1

Write a program that asks a user to input an integer n and m that are number of rows and columns for an array. The program should prompt the user to input $n*m$ integers and store them in two-dimensional array ($n \times m$). Then the program should output the amount of positive numbers in each row.

Input:

2 3
0 -2 3
-5 8 -8

Output:

1 1

Input:

4 5
2 -4 -5 6 7
0 1 -2 9 11
-1 -1 8 3 0
3 4 5 6 7

Output:

3 3 2 5

Practice task 2

Write a program that asks a user to input an integer n and m that are number of rows and columns for an array. The program should prompt the user to input $n*m$ integers and store them in two-dimensional array ($n \times m$). Then the program should output the maximum element of the array.

Input:

4 5
2 -4 -5 6 7
0 1 -2 9 11
-1 -1 8 3 0
3 4 5 6 7

Output:

11

Practice task 3

Write a program that asks a user to input an integer n and m that are number of rows and columns for an array. The program should prompt the user to input n*m integers and store them in two-dimensional array. Then the program should hide negative numbers by x on printing.

Input:

```
4 5
2 -4 -5 6 7
0 1 -2 9 11
-1 -1 8 3 0
3 4 5 6 7
```

Output:

```
2  x  x  6  7
0  1  x  9  11
x  x  8  3  0
3  4  5  6  7
```

Practice task 4

Write a program that asks a user to input an integer n. The program should prompt the user to input n*n integers and store them in two-dimensional array (n x n). Then the program should output all elements of the array in the following pattern.

Input:

```
5
2 -4 -5 6 9
0 1 -2 9 0
-1 -1 8 3 3
3 4 5 6 1
7 11 0 7 12
```

Output:

```
2 -4 -5 6  x
0  1 -2  x  0
-1 -1  x  3  3
3  x  5  6  1
x  11 0  7  12
```

Practice task 5

Write a program that asks a user to input an integer n . The program should prompt the user to input $n*n$ integers and store them in two-dimensional array ($n \times n$). Then the program should the sum of all elements that are not in the main diagonal. (Main diagonal is bolded in input)

Input:

```
3
1 2 3
2 7 4
3 4 6
```

Output:

```
18
```

Input:

```
3
1 2 3
4 5 6
7 8 9
```

Output:

```
30
```

Practice task 6

Write a program that asks a user to input an integer n and m that are number of rows and columns for an array. The program should prompt the user to input $n*m$ integers and store them in two-dimensional array ($n \times m$). Then the program should output the maximum element in each row.

Input:

```
4 5
2 -4 -5 6 7
0 1 -2 9 11
-1 -1 8 3 0
3 4 5 6 7
```

Output:

```
7 11 8 7
```

Practice task 7

Write a program that asks a user to input an integer n . The program should prompt the user to input $n*n$ integers and store them in two-dimensional array ($n \times n$). Then the program should change the elements in the first horizontal half with the second horizontal half.

Input:

```
3
1 2 3
2 7 4
3 4 6
```

Output:

```
3 4 6
2 7 4
1 2 3
```

Input:

```
5
8 9 0 1 5
5 2 3 7 6
0 7 8 9 4
1 2 3 4 5
7 6 4 2 3
```

Output:

```
1 2 3 4 5
7 6 4 2 3
0 7 8 9 4
8 9 0 1 5
5 2 3 7 6
```

Practice task 8

Write a program that asks a user to input an integer n and m that are number of rows and columns for an array. The program should prompt the user to input n*m integers and store them in two-dimensional array (n x m). Then the program should output the minimum element in each column.

Input:

2 3
0 -2 3
-5 8 -8

Output:

5 -2 -8

Input:

4 5
2 -4 -5 6 7
0 1 -2 9 11
-1 -1 8 3 0
3 4 5 6 7

Output:

-1 -4 -5 3 0

Practice task 9

Write a program that asks a user to input an integer n and m. The program should prompt the user to input n*m integers and store them in two-dimensional array (n x m). Then the program should swap a minimum element with a maximum one and print the array.

Input:

3 3
1 2 3
2 7 4
3 4 6

Output:

3 7 6
2 2 4
1 4 3

Practice task 10

Write a program that asks a user to input an integer n and m that are number of rows and columns for an array. The program should prompt the user to input n*m integers, to store them in two-dimensional array (n x m) and an integer k. Then the program should the first k negative elements in the array.

Input:

2 3
0 -2 3
-5 8 -8
2

Output:

-2 -5

Input:

4 5
2 -4 -5 6 7
0 1 -2 9 11
-1 -1 8 3 0
3 4 5 6 7
3

Output:

-4 -5 -2

Practice task 10

Write a program that asks a user to input an integer n. The program should prompt the user to input n*n integers and store them in two-dimensional array (n x n). Then the program should output all elements of the array in ascending order.

Input:

3
0.2 -2.4 3.5
-5.1 8.3 -8.9
1.7 0.5 3.9

Output:

-8.9 -5.1 -2.4
0.2 0.5 1.7
3.5 3.9 8.3

Chapter 6

Strings

string class

As we learned before, a string variable contains a collection of characters surrounded by double quotes. If you want to store a text value, you will use strings. One of the implementations of this task is to use string class. (We also have another way, by using c-styled string, which we will learn later).

main.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4
5     string message = "Hello Programmers";
6     string user = "Ilyas Zhuanyshev";
7
8     string text = user + " said : " + message;
9     cout<<text<<endl;
10
11     return 0;
12 }
```

The output will be:

```
Ilyas Zhuanyshev said : Hello Programmers
```

Strings as a char array

This string can be represented as a one-dimensional array of characters, which is terminated by a null character '\0'. If we create a string variable with some text value, we may print their chars.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4
5     string text = "Sevilla";
6     cout<<text[0]<<endl;
7
8     return 0;
9 }
```

The output will be:

S

It looks like this:

Index	0	1	2	3	4	5	6	7
Variable	S	e	v	i	l	l	a	\n
Address	0x2341	0x2342	0x2343	0x2344	0x2345	0x2346	0x2347	0x2348

Each character of my string is located consequently, forming a char array. Pay attention to the last element. When you create a string variable, it always ends by **\n** symbol (null).

Note that strings can hold numbers as well:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4
5     string text = "2001";
6
7     cout<<text<<endl;
8
9     return 0;
10 }
```

To get the length of a string, we can use the **length()** function.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4
5     string text = "Sevilla";
6
7     cout<<text.length()<<endl;
8
9     return 0;
10 }
```

You can replace all letters 'a' into 'b':

main.cpp

```
1 #include <iostream>
2
3 using namespace std;
4 int main() {
5
6     string text = "sevilla football club de espana";
7
8     for(int i=0;i<text.length();i++){
9         if(text[i]=='a'){
10             text[i] = 'b';
11         }
12     }
13
14     cout<<text<<endl;
15
16     return 0;
17 }
```

The output will be:

sevillb footbbll club de espbnb

Practice works

Strings

Practice task 1

Write a program that asks a user to input a string s. The program should count amount of vowels.

Vowels are: **a, e, i, o, u**

Input:

programmer

Output:

3

Input:

atleticomadrid

Output:

6

Practice task 2

Write a program that asks a user to input a string s. The program should remove all vowels in the string and output it.

Vowels are: **a, e, i, o, u**

Input:

programmer

Output:

prgrmmr

Input:

atleticomadrid

Output:

tltcmdrd

Practice task 3

Write a program that asks a user to input a string **s**. The program should output "Yes" if inputted string is a palindrome, otherwise "No".

Input:

kazzak

Output:

Yes

Input:

attraction

Output:

No

Practice task 4

Write a program that asks a user to input a string **s**. Change all letters of string to upper case.

Input:

Bayern Munich

Output:

BAYERN MUNICH

Input:

Chelsea FC

Output:

CHELSEA FC

Practice task 5

Write a program that asks a user to input a string **s**. Your task is to find out how many times letter 'a' can be found in given sequence. It doesn't matter, upper case or lower.

Input:

FC Arsenal

Output:

2

Input:

Almeria Football Club

Output:

3

Practice task 6

Write a program that asks a user to input a string **s**. Your task is to find out how often each letter from the alphabet is found in the string. It does not matter, upper case or lower. Result must be in alphabetic order.

Input:

Assets

Output:

A - 1

E - 1

S - 3

T - 1

Input:

Programming

Output:

A - 1

G - 2

I - 1

M - 2

N - 1

O - 1

P - 1

R - 2

Chapter 7

Functions and arguments

Functions (return type and void)

A function is a block of code that allows to structure programs in segments of code to perform individual tasks. Imagine that we are going to print different 5 texts 10 times. To complete this task, you may use simple loops for each text.

main.cpp

```
1 #include <iostream>
2
3 using namespace std;
4 int main() {
5
6     string t1 = "Real Madrid";
7     string t2 = "Barcelona";
8     string t3 = "Chelsea";
9     string t4 = "Liverpool";
10    string t5 = "Juventus";
11
12    for(int i=0;i<10;i++){
13        cout<<t1<<endl;
14    }
15
16    for(int i=0;i<10;i++){
17        cout<<t2<<endl;
18    }
19
20    // Then continue for other strings
21
22    return 0;
23 }
```

Ok, printing text 10 times is not difficult. But, what about finding the geometric mean of double array? What about sorting array and removing maximum and minimum elements for 20 integer arrays? Will we repeat codes for implementation our tasks? No. Of course, it is better to create a function, which performs this procedure, and call this function for each array. In real life, there are too many complex algorithms, which contains more than 1000 lines of code. That is why we need to create a function make our life better. The same logic for previous task. We will create a simple function, that prints a text value 10 times, and call this function for each 5 texts separately.

main.cpp

```
1 #include <iostream>
2
3 using namespace std;
4
5 void printText(string text){
6     for(int i=0;i<10;i++){
7         cout<<text<<endl;
8     }
9 }
10
11 int main(){
12
13     string t1 = "Real Madrid";
14     string t2 = "Barcelona";
15     string t3 = "Chelsea";
16     string t4 = "Liverpool";
17     string t5 = "Juventus";
18
19     printText(t1);
20     printText(t2);
21     printText(t3);
22     printText(t4);
23     printText(t5);
24
25     return 0;
26 }
```

As you see from example, we created a function, called **printText()**, that takes as a parameter string variable - **text**, and prints **this** text 10 times by using loop. I reuse this function for each string variable (t1, t1, t3, t4, t5), in **main()** function.

You can divide your code into separate functions. How you divides your code among different functions is up to you, but logically the division usually is such that each function performs a specific task.

Code syntax:

```
void someFunction(parameters) {
    //my code
}
```

someFunction() - is a name of the function

void - means that the function does not have a return value. Functions can return some data type or return void, which means return nothing.

Arguments

Function can have parameters. We also call them - arguments. Parameters can be any data type. You declare them as a local variable. A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

Let's try to look at another example, which returns sum of three parameters.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int getSum(int a, int b, int c) {
5     return a+b+c;
6 }
7
8 int main() {
9
10     int x = getSum(1,2,3);
11     int y = getSum(5,6,7);
12     int z = getSum(10,20,30);
13
14     cout<<getSum(x,y,z)<<endl;
15
16     return 0;
17 }
```

In this example, we created a function, called **getSum(int a, int b, int c)** with 3 parameters. Instead of void, we wrote **int**. It means the result of my function will be an integer value. In the end, the function returns us their sum. Pay attention to the calling of this function. We created an integer **x = getSum(1,2,3)**, it means that we set function parameters as **a = 1, b = 2, c = 3**. Function returns 6, because **a+b+c = 6**. Finally, value of **int x** will be **6, y = 18** and **z = 60**. The output of program will be - 84.

Arguments can be also an array variable. You can create a function that takes an array and its length as parameters. Let's look at the example, given below. We have to find the maximum element in array, by using function.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int getMax(int arr[], int size){
5
6     int max = arr[0];
7
8     for(int i=1;i<size;i++){
9         if(arr[i]>max){
10             max = arr[i];
11         }
12     }
13     return max;
14 }
15
16 int main(){
17
18     int massiv1[] = {10,20,30,40,50};
19     int massiv2[] = {-1,100,-59,23,2};
20
21     cout<<getMax(massiv1, 5)<<endl;
22     cout<<getMax(massiv2, 5)<<endl;
23
24     return 0;
25 }
```

We created a function, with arguments **int arr[]** and **int size**. Function uses classic algorithm of finding maximum element, and then returns it.

Function prototype

You may declare function and implement it later. A function declaration tells the compiler about a function name and how to call the function. We also call them – **function prototypes**. The actual body of the function can be defined separately. For example, declaration of our previous function will be like this:

```
int getMax(int[], int);
```

Parameter names are not important in function declaration only their type is required.

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

The full code will be like this:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int getMax(int[], int);
5
6 int main() {
7
8     int massiv1[] = {10,20,30,40,50};
9     int massiv2[] = {-1,100,-59,23,2};
10
11     cout<<getMax(massiv1, 5)<<endl;
12     cout<<getMax(massiv2, 5)<<endl;
13
14     return 0;
15 }
16
17 int getMax(int arr[5], int size){
18
19     int max = arr[0];
20
21     for(int i=1;i<size;i++){
22         if(arr[i]>max){
23             max = arr[i];
24         }
25     }
26     return max;
27 }
```

Function overloading

Function overloading is a feature of C++ that allows us to create multiple functions with the same name, so long as they have different parameters.

```
int getSum(int a, int b){
    return a+b;
}
int getSum(int a, int b, int c){
    return a+b+c;
}
int getSum(int a, int b, int c, int d){
    return a+b+c+d;
}
```

In this code, you see that all functions are different, according to parameters, but with same name. It is very important to declare functions with different combinations of parameters, because when you want to invoke your function, compiler must know, which exactly function you want to call.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int getSum(int a, int b) {
5     return a+b;
6 }
7
8 int getSum(int a, int b, int c) {
9     return a+b+c;
10 }
11
12 int getSum(int a, int b, int c, int d) {
13     return a+b+c+d;
14 }
15
16 int main() {
17
18     cout<<getSum(1,2)<<endl;
19     cout<<getSum(1,2,3)<<endl;
20     cout<<getSum(1,2,3,4)<<endl;
21
22     return 0;
23 }
```

In this example, you see that line 18 calls first function **getSum(int a, int b)**, because we used only 2 integer parameters. The same logic for line 20, where we invoke a function with 4 arguments.

Default values for parameters

When you define a function, you can specify a default value for each of the last parameters. This value will be used if the corresponding argument is left blank when calling to the function.

```
int getSum(int a, int b=10) {
    return a+b;
}
```

This is done by using the assignment operator and assigning values for the arguments in the function definition.

If a value for that parameter is not passed when the function is called, the default given value is used, but if a value is specified, this default value is ignored and the passed value is used instead.

Let's look at this example:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int getSum(int a, int b=10) {
5     return a+b;
6 }
7
8 int main() {
9
10     cout<<getSum(1)<<endl;
11     cout<<getSum(1,2)<<endl;
12
13     return 0;
14 }
```

The output will be:

```
11
3
```

As you see from output, the first line shows 11, because on line 10 we invoked our function, but only with one argument. The function **getSum()** takes parameter **int b** as 10, and returns 11. On line 11 we invoke our function with 2 arguments, **a = 1** and **b = 2**. That is why our result will be 3.

Be careful on using function overloading and taking default parameter. Imagine that, you created 3 functions for getting sum.

```
int getSum(int a) {
    return a;
}
int getSum(int a, int b=10) {
    return a+b;
}
int getSum(int a, int b, int c) {
    return a+b+c;
}
```

The first function takes only 1 parameter, but the second one can take 1 or 2. Here, you cannot invoke function with 1 parameter. Compiler doesn't know which function to call, first or second. We have 2 candidates.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int getSum(int a) {
5     return a;
6 }
7
8 int getSum(int a, int b=10) {
9     return a+b;
10 }
11
12 int getSum(int a, int b, int c) {
13     return a+b+c;
14 }
15
16 int main() {
17
18     cout<<getSum(1)<<endl;
19     cout<<getSum(1,2)<<endl;
20     cout<<getSum(1,2,3)<<endl;
21
22     return 0;
23 }
```

On line 18, we invoke function **getSum(2)**. Candidates can be **getSum** with 1 parameter and **getSum** with 2 parameters. The compiler will return an error and will execute it.

Be careful on using it. The best solution is to remove first function and invoke second function for 1 integer and 2 integer parameters. That is all.

Header files

As programs grow larger, it becomes increasingly difficult to declare all functions in one **cpp** file and use them. Wouldn't it be nice if you could put all your forward declarations in one place and then import them when you need them? C++ code files with **cpp** extension are not the only files commonly seen in C++ programs. The other type of file is called a header file. Header files usually have **.h** extension. Let's create a header file, called **myFunctions.h**, where we will store our functions.

myFunctions.h

```
1 #include <iostream>
2 using namespace std;
3
4 int sum(int a, int b){
5     return a+b;
6 }
7
8 int printText(string text, int amount){
9     for(int i=0;i<amount;i++){
10         cout<<text<<endl;
11     }
12 }
```

main.cpp

```
1 #include <iostream>
2 #include "myfunctions.h"
3
4 using namespace std;
5
6 int main(){
7
8     cout<<sum(10,20)<<endl;
9
10    printText("Ilyas", 5);
11
12    return 0;
13 }
```

In this example, you see that we divided our code into separate files. Then, the main file includes my header file and use functions in it. In future, it will be easy and conveniently to separate your functions, algorithms in different files.

Practice works

Functions and arguments

Practice task 1

Write a function that accepts as an argument two numbers and returns percentage of the first number by the second.

Input:

3 6

Output:

50%

Input:

14 7

Output:

200%

Practice task 2

Write a function that accepts as an argument a word and return the number of vowels. (Vowels: a, e, i, o, u)

Input:

Ernis is playing with cat

Output:

7

Input:

Yerzhan broke his Play Station

Output:

9

Practice task 3

Write a function that accepts as an argument two dimensional array, number of rows and number of columns of the array. Program should count an amount of even numbers.

Input:

3 3
1 2 3
4 5 6
7 8 9

Output:

4

Input:

3 3
8 5 4
0 0 9
1 0 6

Output:

6

Practice task 4

Write a function that accepts as an argument a word and outputs the word without vowels in it. (Vowels: a, e, i, o, u)

Input:

Douglas Junior

Output:

Dgls Jnr

Input:

Abay scored goal

Output:

by scrd gl

Practice task 5

Write a function that accepts as an argument two dimensional array, number of rows and number of columns of the array. The function should remove the column where the maximum element is located.

Input:

```
3 3
8 5 4
0 0 9
1 0 6
```

Output:

```
8 5
0 0
1 0
```

Input:

```
3 4
-4 -5 -5 5
8 4 -7 6
-6 -8 9 4
```

Output:

```
-4 -5 5
8 4 6
-6 -8 4
```

Practice task 6

Write a function that accepts as arguments a word and a letter. The function returns the number of occurrence of the letter in the word.

Input:

```
Leo Santana
a
```

Output:

```
3
```

Practice task 7

Write a function that accepts as an argument two dimensional array, number of rows and number of columns of the array. The function should return sum of elements, which have odd indexes by row and column.

Input:

3 4

-4 -5 -5 5

8 4 -7 6

-6 -8 9 4

Output:

10

Input:

5 5

-1 5 -5 7 9

3 8 4 -7 9

3 -6 -8 9 4

8 -6 -8 4 4

9 2 7 9 9

Output:

-1

Practice task 8

Write a function that accepts as an argument an array. The function should delete all elements before the minimum element.

Input:

10

9 7 10 -3 8 5 3 4 0 8

Output:

8 5 3 4 0 8

Practice task 9

Write a function that accepts as an argument two dimensional array, number of rows and number of columns of the array. The function should return sum of positive integers which are located in the rows that does not contain zero elements.

Input:

3 3
8 5 4
0 0 9
1 0 6

Output:

17

Input:

3 4
-4 -5 0 5
8 1 -7 6
-1 -8 9 4

Output:

28

Practice task 10

Write a function that accepts as an argument two dimensional array, number of rows and number of columns of the array. The function should output indexes of rows where at least one zero element is located. Assume that index starts from 1.

Input:

3 3
8 5 4
0 0 9
1 0 6

Output:

2
3

Chapter 8

Pointers & references

Pointers

When our program instantiates a variable, a free memory address is automatically assigned to the variable, and any value we assign to the variable is stored in this memory address. For example, let's say that an integer **int num = 100**, is assigned memory location **0x28ff1c**. Whenever the program sees the variable **num** in an expression or statement, it knows that it should look in memory location **0x28ff1c** to get the value.

To get the address of a variable, we use special symbol, called **address-of operator (&)** or as we know - **ampersand**. We also call it reference.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int num = 100;
7     cout<<&num<<endl;
8
9     return 0;
10 }
```

The output will be (in my computer):

0x28ff1c

As we understand, that the address value of any variable can be taken by using **&** symbol.

Pointer - is a variable whose value is the address of another variable. In the other words, we can say that **pointer is a variable that stores the address of another variable**. Pointers are a very powerful feature of the C++ language that has many uses in lower level programming. It is something like a remote control, for variable. You can manipulate with value of another variable, by using pointers.

To create a pointer, we use special symbol, called - **dereference operator (*)** or asterisk.

```
int *ptr;
```

Like any variable or constant, you must declare a pointer before you can work with it.

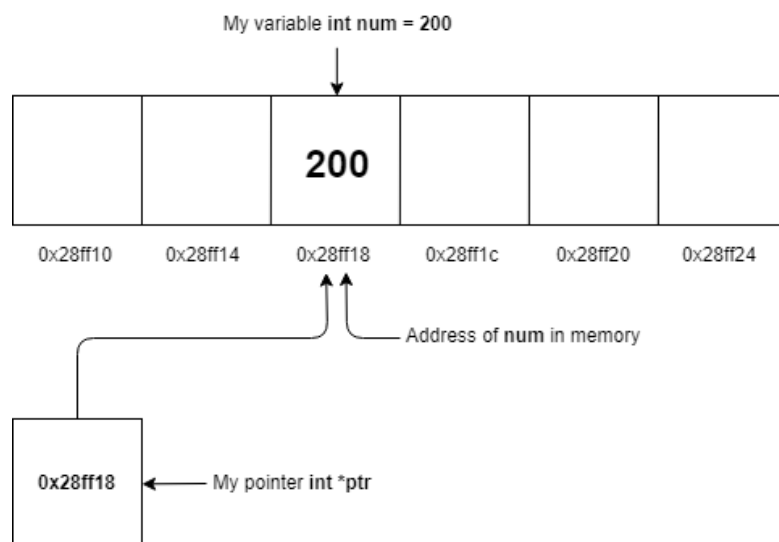
Let's look at the example, given below:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int num = 200;
7     int *ptr = &num;
8
9     cout<<num<<endl;
10    cout<<&num<<endl;
11    cout<<ptr<<endl;
12    cout<<*ptr<<endl;
13
14    return 0;
15 }
```

The output will be:

```
200
0x28ff18
0x28ff18
200
```



The picture above illustrates how in fact we create an integer variable and pointer to this variable.

What does it mean? Firstly, when you declare your pointer **int*ptr**, you always use asterisk operator (*). You cannot for example just say that **int*ptr = 200**, because **int *ptr** needs an address value. You must assign to **int*ptr** an address value of any integer, declared before. That is why, we firstly declare a variable, **int num = 200**, and then we assign this integer variable's address into our pointer **int *ptr = &num**.

```
int num = 200;
int *ptr = &num;
```

It is very important, be careful on assigning pointer.

You may also just declare a pointer variable, and then, assign integer's address to this pointer.

```
int *ptr;
int num = 200;
ptr = &num;
```

Is it clear? Okay. Let's try to understand, that when we declare a pointer, we use asterisk operator (*), but when we use already declared pointer, we don't use asterisk operator (*). **If we use asterisk operator (*) for already declared pointer, we return the value of variable, located in this address.** That is why the result of **cout<<*ptr<<endl** is **200**, the value of **int num** will be also **200**.

To print the address value of pointer, just write:

```
cout<<ptr<<endl;
```

Now, let's try to change the value of **int num = 200**, by using pointer.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int num = 200;
7     int *ptr = &num;
8
9     *ptr=*ptr+1; // You cannot write *ptr++
10
11     cout<<num<<endl;
12     cout<<*ptr<<endl;
13
14     return 0;
15 }
```

The output will be:

```
201
201
```

What happened here? By using, asterisk operator (*), we changed the value that is located on address **0x28ff18** by one. It means, our variable **int num = 200** will be changed to **201**. Then, we print the value of **num** and the value that is located on address **0x28ff18**. In the other words, ***ptr** returns the value of **num**. Notice that we wrote on comments that we cannot write ***ptr++** to increment the value of **num** by 1. Why? Because ***ptr++** is equivalent to ***(ptr++)**, it means we **increment the address of our pointer** by one, by losing control over variable **num**. The correct way is to write **(*ptr)++**, which means we will increment the value of variable **num** by 1.

Now, let's look at the next example, to understand pointers better.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int num = 200;
7     int *ptr = &num;
8     int anotherNum = *ptr;
9
10    *ptr = 500;
11
12    cout<<num<<endl;
13    cout<<*ptr<<endl;
14    cout<<anotherNum<<endl;
15
16    return 0;
17 }
```

The output will be:

```
500
500
200
```

We created 2 variables, **num** and **anotherNum**. The value of **num** is 200. We created a pointer **int *ptr** to our variable **num**. The value of **anotherNum** is the value (not address) of a pointer **ptr** is 200. We changed the value of a pointer **ptr** to 500, it means that the value of **num** will be 500. But, the address value of **anotherNum** is different, because pointer **ptr** doesn't point to it. In the end, the value of **num** will be 500, and the value of **anotherNum** will be 200.

Practice works

Pointers & references

Practice task 1

Write a function that accepts as an argument two numbers and returns percentage of the first number by the second.

Input:

3 6

Output:

50%

Input:

14 7

Output:

200%

Chapter 9

Pointers as arrays (Dynamic arrays)

Dynamic arrays

Pointers can be used as an array. In fact, pointers and arrays are interchangeable in many cases. For example, a pointer that points to the beginning of an array can access that array by using either pointer arithmetic or array-style indexing. Consider the following situation:

main.cpp

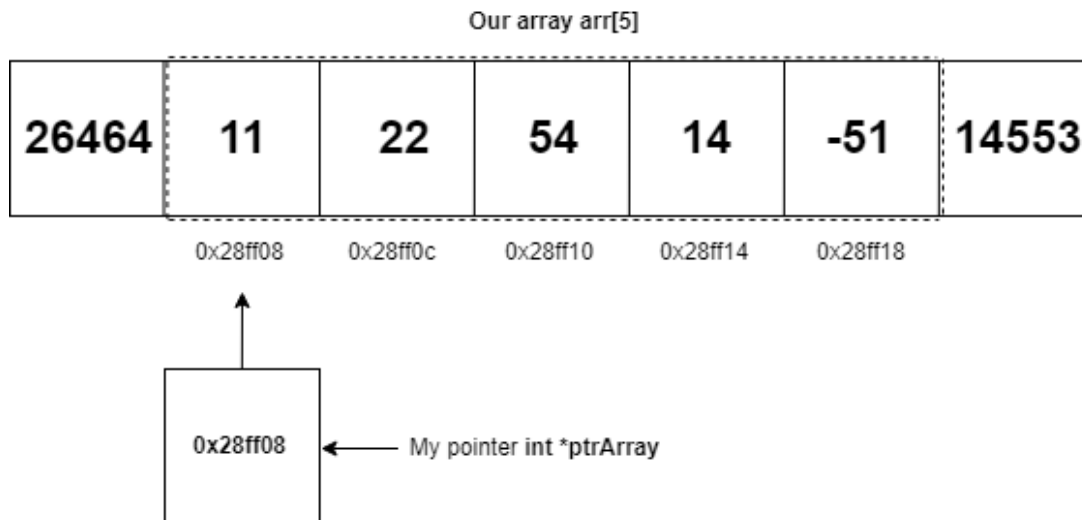
```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int arr[5] = {11,22,54,14,-51};
7     int *ptrArray;
8
9     ptrArray = arr;
10
11     cout<<*ptrArray<<endl;
12     cout<<*(ptrArray+1)<<endl;
13     cout<<*(ptrArray+2)<<endl;
14
15     return 0;
16 }
```

The output will be:

```
11
22
54
```

What happened? We declared one dimensional integer array with values {11,22,54,14,-51}, and created a pointer **int *ptrArray**. After that, we equated array to pointer. In fact, we equated address of first element of array **arr** to our pointer **ptrArray**. In the other words, we can say that pointers and arrays support the same set of operations, with the same meaning for both. The main difference being that pointers can be assigned new addresses, while arrays cannot. In example above, when we added 1 to our pointer **ptrArray**, we pointed to the next element, located in next address. When we normally create one dimensional array, all elements of array are located one after another.

It looks like this:



Two numbers (26464, 14553), that are located outside of array are default random numbers in memory. When you create an empty array, without assigning elements, you take default random numbers from memory as a value.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int arr[5]; // We didn't assign elements
7
8     for(int i=0;i<5;i++){
9         cout<<arr[i]<<endl;
10    }
11
12    return 0;
13 }
```

The output will be: (Different on different devices)

```
-2
1975664954
1975666125
4200912
2686816
```

Here you see, if you don't assign values for each index, you will get random memory numbers as a values of each index.

Finally, we can use pointers as arrays.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int *arr = new int[1000];
7     int n;
8     cin>>n;
9
10    for(int i=0;i<n;i++){
11        cin>>*(arr+i);
12    }
13
14    for(int i=0;i<n;i++){
15        cout<<*(arr+i)<<" ";
16    }
17
18    return 0;
19 }
```

In this example, created a pointer by **allocating** (reserving) 1000 elements. Then, as usual, we insert an amount of elements we want to insert. Then we insert elements and print them. Pay attention to that how did we implement insertion of all elements into our dynamic array. In every iteration, we pass to the next memory space, by adding integer *i* to our pointer. Because, pointer points to the 0th element of our dynamic array.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int *arr = new int[1000]; //similar to int arr[1000];
7     *arr = 55; // similar to arr[0] = 55;
8     *(arr+1) = 77; // similar to arr[1] = 77;
9     *(arr+2) = 105; // similar to arr[2] = 105;
10
11    cout<<arr[0]<<endl; // you may also use like this
12    cout<<arr[1]<<endl;
13    cout<<*(arr+2)<<endl;
14
15    return 0;
16 }
```

The output will be:

```
55
77
105
```

Finally, can use pointers as arrays, or call them - dynamic array.

Deallocating memory by "delete" operator

The operator **delete** is used to destroy array and non-array (pointer) objects which are created by **new** expression, by deallocating memory from heap.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int *arr = new int[5];
7     arr[0] = 10;
8     arr[1] = 20;
9     arr[2] = 30;
10    arr[3] = 40;
11    arr[4] = 50;
12
13    for(int i=0;i<5;i++){
14        cout<<arr[i]<<" ";
15    }
16
17    delete[] arr;
18
19    return 0;
20 }
```

In this case, pointer to object will not be destroyed. The value or memory block pointed by pointer is destroyed.

c-style string symbolic constant

Pointers can be used to access a variable by its address, and this access may include modifying the value pointed. But it is also possible to declare pointers that can access the pointed value to read it, but not to modify it. For this, it is enough with qualifying the type pointed to by the pointer as **const**.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     int a = 100;
7     int b = 200;
8     const int *ptr = &a;
9     int c = *ptr; //Everything is ok, because read-only
10    cout<<c<<endl;
11    *ptr = b;
12    cout<<*ptr<<endl; //You cannot modify value of constant pointer
13
14    return 0;
15 }
```

The compiler will return an error:

error: assignment of read-only location '* ptr'

As we told in previous chapters about strings, string literals are arrays of chars, which contains characters plus the terminating null (**'\0'**) character. We can create constant pointer of chars, with a value of string.

```
const char *name = "Ilyas";
const char *surname = "Zhuanyshev";
```

You cannot directly equalize string values to non-constant pointers of char.

```
char *name = "Ilyas";
char *surname = "Zhuanyshev";
```

So, let's compare **c-styled string by using char array** vs. **c-styled string symbolic constant**.

c-styled by array

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     char name[] = "Ilyas";
7     cout<<name;
8
9     return 0;
10 }
```

c-styled string symbolic constant

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     const char *name = "Ilyas";
7     cout<<name<<endl;
8
9     return 0;
10 }
```

While these above two programs operate and produce the same results, C++ deals with the memory allocation for these slightly differently.

In the fixed array case, the program allocates memory for a fixed array of length 5, and initializes that memory with the string **"Ilyas\0"**. Because memory has been specifically allocated for the array, you are free to change the contents of the array.

In the symbolic constant case, how the compiler handles this is implementation defined. What usually happens is that the compiler places the string **"Ilyas\0"** into read-only memory somewhere, and then sets the pointer to point to it. For optimization purposes, multiple string literals may be consolidated into a single value.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     const char *name = "Ilyas";
7     const char *name2 = "Ilyas";
8
9     return 0;
10 }
```

These are two different string literals with the same value. The compiler may opt to combine these into a single shared string literal, with both `name1` and `name2` pointed at the same address. Thus, if `name1` was not `const`, making a change to `name` could also impact `name2`.

Feel free to use c-style string symbolic constants if you need read-only strings in your program, but always make them `const`.

You also may use pointers of `char` as a char array for inserting data from console:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6     char *name = new char[100];
7     cin>>name;
8
9     for(int i=0;name[i]!='\0';i++) {
10         cout<<name[i]<<endl;
11     }
12
13     return 0;
14 }
```

In this example, we don't use constant char pointers, because we dynamically change content.

Practice works

Pointers as arrays (Dynamic arrays)

Practice task 1

Write a program that converts given sequence by the following rules (use pointers):

- 1 - If the letter is in upper case make it lower case;
- 2 - If the letter is in lower case make it upper case;
- 3 - If it's not a letter left it as is;

Input:

eRNIS aND eRZHAN aRE pLAYING fifa 2019

Output:

Ernis And Erzhan Are Playing FIFA 2019

Input:

CONTESTS

Output:

contests

Practice task 2

Write a program that asks user to input a sentence. Then user inputs a text to replace and another text as a value. Your program must replace text in sentence to another.

Input:

Erzhan played FIFA19 against Ernis. Erzhan became a winner.

Erzhan

Abay

Output:

Abay played FIFA19 against Ernis. Abay became a winner.

Input:

Abay played FIFA19 against Ernis. Abay became a winner.

FIFA19

PES20

Output:

Abay played PES20 against Ernis. Abay became a winner.

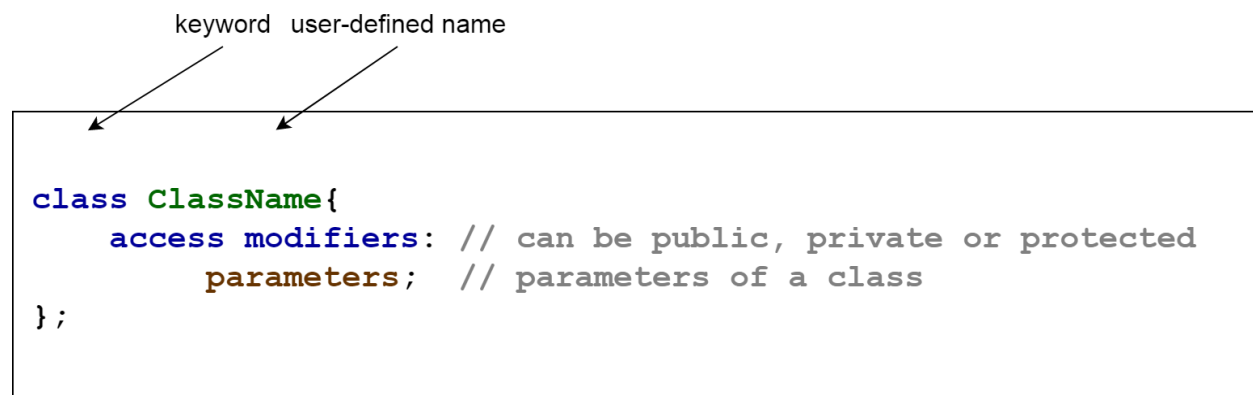
Chapter 10

Classes and Objects

Classes

Classes are the central feature of C++ that supports object-oriented programming. In other words, classes are blueprints, which used to specify the form of an object. In real life, we can get an analogy with human. Human is a prototype, class, with special attributes, such as name, surname, age, weight, height, nationality etc. Here, **people are objects of class Human**. There are **7 billion objects** of class **Human** around the World. Some of them taller, some of them younger, most of them speaks in different languages etc. If we take as example, one of the most popular singers, such as **Taylor Swift**, she is an object of class **Human**. Her **height** is 178 cm, **birthdate** – 13 December 1989, **native language** - English etc.

In our life, everything are object of some classes. Computers, mobile phones, tables, desks, cups, bags, shoes etc. Let's say that I have a car, Toyota Camry, silver colored, manufactured in 2012, bought in Toyota Motors Almaty. My Toyota Camry is an object of class Car. In our city, we have about 500000 cars, which means that we have about 500000 objects of a class Car. My friend drives his Mercedes E230, blue colored and manufactured in 2010. We have same attributes with different values. Because all cars has in general same attributes. Mine is silver colored, my friend's – blue. This is a difference of properties (attribute values) between our **objects**.



```
class ClassName{
    access modifiers: // can be public, private or protected
    parameters; // parameters of a class
};
```

Okay, now, let's try to create a class in C++.

We will call it Car, with four parameters: name, year, color, engine volume.

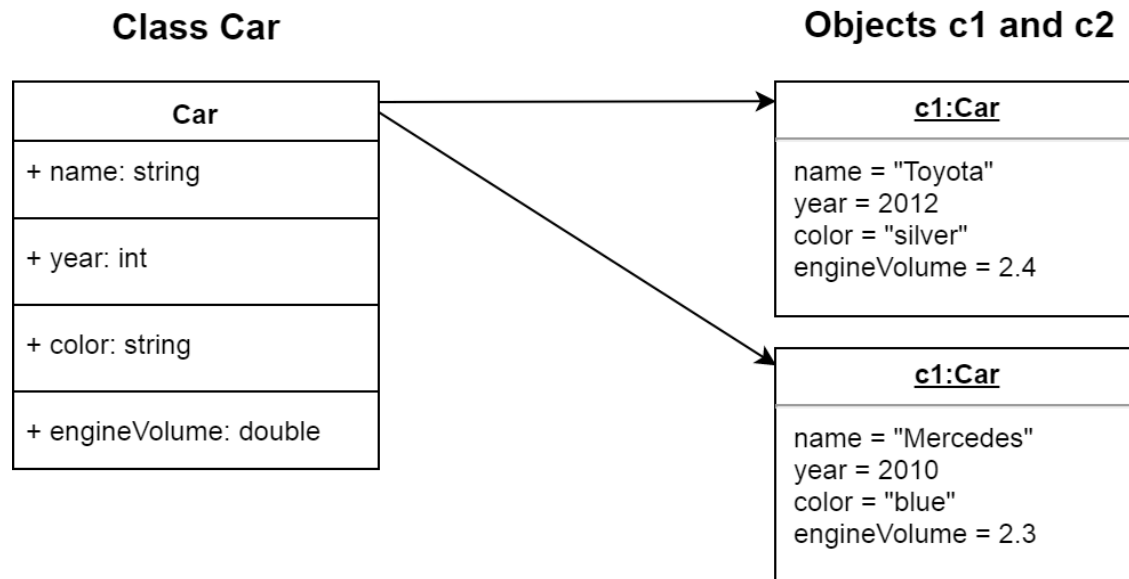
main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 class Car{
5
6     public:
7         string name;
8         int year;
9         string color;
10        double engineVolume;
11
12 };
13
14 int main() {
15
16     Car c1, c2;
17     c1.name = "Toyota";
18     c1.year = 2012;
19     c1.color = "silver";
20     c1.engineVolume = 2.4;
21
22     c2.name = "Mercedes";
23     c2.year = 2010;
24     c2.color = "blue";
25     c2.engineVolume = 2.3;
26
27     cout<<"My car is "<<c1.color<<" colored "<<c1.name;
28     cout<<", from "<<c1.year;
29     cout<<", with engine volume "<<c1.engineVolume<<" liters"<<endl;
30
31     cout<<"Your car is "<<c2.color<<" colored "<<c2.name;
32     cout<<", from "<<c2.year;
33     cout<<", with engine volume "<<c2.engineVolume<<" liters"<<endl;
34
35     return 0;
36 }
```

The output will be:

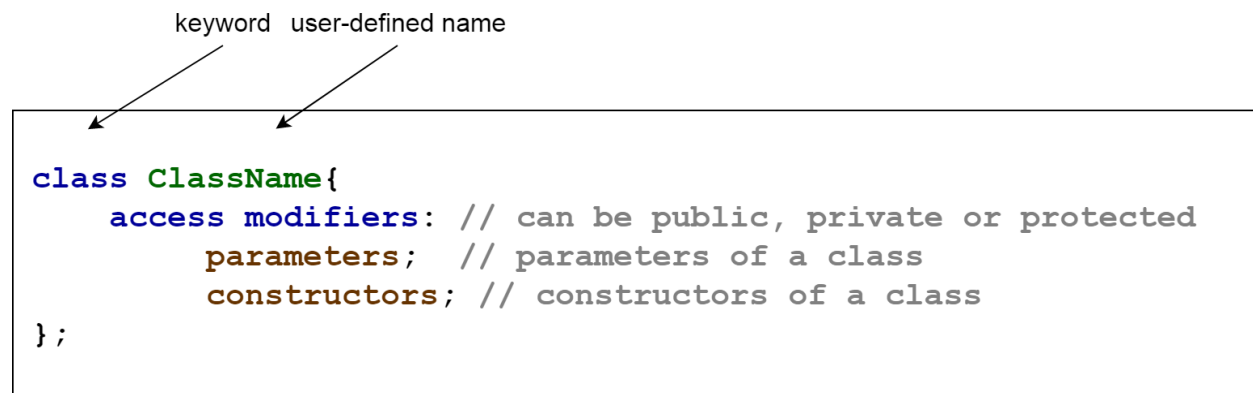
```
My car is silver colored Toyota, from 2012, with engine volume 2.4 liters
Your car is blue colored Mercedes, from 2010, with engine volume 2.3 liters
```

As you see from results, we created a class, called Car with four parameters. We used **public** access-modifier, to declare parameters. We declared 2 objects, c1 and c2, then assigned their parameters. In the end, then we printed data of our objects separately.



Constructors

Constructors are special class members, which called by the compiler every time an object of that class is instantiated. We can use constructors to initialize and set object parameters. Constructors have the same name as the class and may be defined inside or outside the class definition.



You can create more than one constructor, but with different arguments.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 class Car{
5
6     public:
7         string name;
8         int year;
9         string color;
10        double engineVolume;
11
12    Car(){
13        name = "No Name";
14        year = 2000;
15        color = "No Color";
16        engineVolume = 0;
17    }
18
19    Car(string name, int year, string color, double engineVolume){
20        this->name = name;
21        this->year = year;
22        this->color = color;
23        this->engineVolume = engineVolume;
24    }
25
26 };
27
28 int main(){
29
30     Car c1("Toyota", 2012, "silver", 2.4);
31     Car c2("Mercedes", 2010, "blue", 2.3);
32     Car c3;
33
34     cout<<"My car is "<<c1.color<<" colored "<<c1.name;
35     cout<<" , from "<<c1.year;
36     cout<<" , with enigne volume "<<c1.engineVolume<<" liters"<<endl;
37
38     cout<<"Your car is "<<c2.color<<" colored "<<c2.name;
39     cout<<" , from "<<c2.year;
40     cout<<" , with enigne volume "<<c2.engineVolume<<" liters"<<endl;
41
42     cout<<"His car is "<<c3.color<<" colored "<<c3.name;
43     cout<<" , from "<<c3.year;
44     cout<<" , with enigne volume "<<c3.engineVolume<<" liters"<<endl;
45
46     return 0;
47 }
```

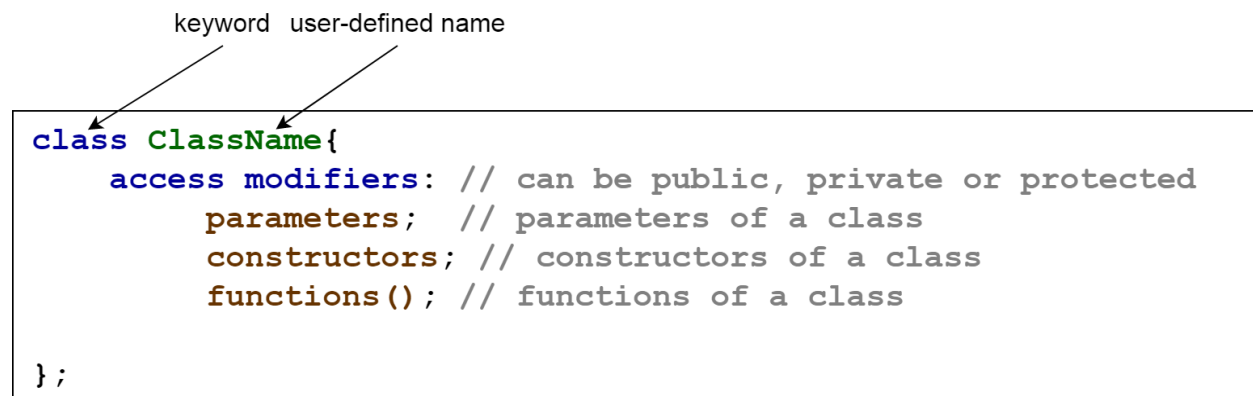
The output will be:

```
My car is silver colored Toyota, from 2012, with enigne volume 2.4 liters
Your car is blue colored Mercedes, from 2010, with enigne volume 2.3 liters
His car is No Color colored No Name, from 2000, with enigne volume 0 liters
```

From this code, you see that we created two constructors. First one is default (with no arguments), the second one - parameterized (with arguments). Look carefully at the second constructor. We specify argument names same as parameter names. To avoid a conflict between names, we use special keyword, called - **this**. The keyword **this** is used to refer to the current instance of the class. When we write **this->color = color**, it means the **string color**, which is parameter of current class is equal to the argument variable color. Now, according to this constructors, we can create an instance of a class Car by setting parameters right away. If we write **Car c1("Toyota", 2012, "silver", 2.4)**, we will create an object of a class Car with already set parameters, taken from constructor arguments.

Functions

Classes can have functions. We also call them – member functions. Member functions are behaviors of our class.



```
class ClassName{
    access modifiers: // can be public, private or protected
    parameters; // parameters of a class
    constructors; // constructors of a class
    functions(); // functions of a class
};
```

Functions can be void or return type (int, double, string, boolean etc). You can create two or more functions with same names but with different arguments, by overloading it.

We can declare a prototype of a function, and then implement it outside of our class. Second way is to declare and implement our constructor immediately inside of our class.

Let's say that our car has have a functions **void ride()**.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 class Car{
5     public:
6         string name;
7         int year;
8         string color;
9         double engineVolume;
10
11     Car(){
12         name = "No Name";
13         year = 2000;
14         color = "No Color";
15         engineVolume = 0;
16     }
17     Car(string name, int year, string color, double engineVolume){
18         this->name = name;
19         this->year = year;
20         this->color = color;
21         this->engineVolume = engineVolume;
22     }
23     void ride();
24
25 };
26 void Car::ride(){
27     cout<<"Car "<<name<<" "<<color<<" colored";
28     cout<<", from "<<year;
29     cout<<", with volume "<<engineVolume<<" liters is riding"<<endl;
30 }
31 int main(){
32
33     Car c1("Toyota", 2012, "silver", 2.4);
34     Car c2("Mercedes", 2010, "blue", 2.3);
35     Car c3;
36
37     c1.ride();
38     c2.ride();
39     c3.ride();
40
41     return 0;
42 }
```

The output will be:

```
Car Toyota silver colored, from 2012, with volume 2.4 liters is riding
Car Mercedes blue colored, from 2010, with volume 2.3 liters is riding
Car No Name No Color colored, from 2000, with volume 0 liters is riding
```

We can use headers, to store classes in separate files.

car.h

```
1 #include <iostream>
2 using namespace std;
3
4 class Car{
5     public:
6         string name;
7         int year;
8         string color;
9         double engineVolume;
10
11     Car(){
12         name = "No Name";
13         year = 2000;
14         color = "No Color";
15         engineVolume = 0;
16     }
17     Car(string name, int year, string color, double engineVolume){
18         this->name = name;
19         this->year = year;
20         this->color = color;
21         this->engineVolume = engineVolume;
22     }
23     void ride();
24
25 };
26 void Car::ride(){
27     cout<<"Car "<<name<<" "<<color<<" colored";
28     cout<<" from "<<year;
29     cout<<" with volume "<<engineVolume<<" liters is riding"<<endl;
30 }
```

main.cpp

```
1 #include <iostream>
2 #include "car.h"
3 using namespace std;
4
5 int main(){
6
7     Car c1("Toyota", 2012, "silver", 2.4);
8     Car c2("Mercedes", 2010, "blue", 2.3);
9     Car c3;
10
11     Car cars[] = {c1,c2,c3};
12     for(int i=0;i<3;i++){
13         cars[i].ride();
14     }
15
16     return 0;
```



```
17 }
```

The output will be:

```
Car Toyota silver colored, from 2012, with volume 2.4 liters is riding
Car Mercedes blue colored, from 2010, with volume 2.3 liters is riding
Car No Name No Color colored, from 2000, with volume 0 liters is riding
```

In this example, we see that we divided our class into two files (header file and cpp file). In cpp file, we created an array of class Car to store all objects. By using loop, we can call each object's function – **ride()**, which prints data of a car.

By using classes and object, you may create own complex data type, which has own parameters and functions, to reuse them repeatedly.

Now, let me show you an example of application for managing with students. You can add student and list students by using console menu.

Firstly, let's create a student class, with parameters: id, name, surname, group, GPA.

student.h

```
1 #include <iostream>
2 using namespace std;
3
4 class Student{
5     public:
6         int id;
7         string name;
8         string surname;
9         string group;
10
11     Student(){
12         this->id = 0;
13         this->name = "----";
14         this->surname = "----";
15         this->group = "----";
16     }
17     Student(int id, string name, string surname, string group){
18         this->id = id;
19         this->name = name;
20         this->surname = surname;
21         this->group = group;
22     }
23     void printData();
24 };
25 void Student::ride(){
26     cout<<id<<" - "<<name<<" "<<surname<<" - "<<group<<endl;
27 }
```

Now, we can create our console menu and do operations with a student list.

main.cpp

```
1 #include <iostream>
2 #include "student.h"
3 using namespace std;
4
5 int main() {
6
7     Student memory[10000];
8     int current = 0, id = 1;
9     string choice, name, surname, group;
10
11     while(true) {
12
13         cout<<"PRESS 1 TO ADD STUDENT"<<endl;
14         cout<<"PRESS 2 TO LIST STUDENTS"<<endl;
15         cout<<"PRESS 0 EXIT"<<endl;
16
17         cin>>choice;
18
19         if(choice=="1") {
20
21             cout<<"Insert name:"<<endl;
22             cin>>name;
23             cout<<"Insert surname:"<<endl;
24             cin>>surname;
25             cout<<"Insert group:"<<endl;
26             cin>>group;
27             Student st(id, name, surname, group);
28             memory[current] = st;
29             current++;
30             id++;
31
32         }else if(choice=="2") {
33
34             for(int i=0;i<current;i++) {
35                 memory[i].printData();
36             }
37
38         }else if(choice=="0") {
39             cout<<"Good Bye!"<<endl;
40             break;
41         }else{
42             cout<<"Wrong command, try again"<<endl;
43         }
44     }
45
46     return 0;
47 }
```

Let's test it:

```
PRESS 1 TO ADD STUDENT
PRESS 2 TO LIST STUDENTS
PRESS 0 EXIT
```

1

Insert name:
Ilyas

Insert surname:
Zhuanyshev

Insert group:
IT01

```
PRESS 1 TO ADD STUDENT
PRESS 2 TO LIST STUDENTS
PRESS 0 EXIT
```

1

Insert name:
Erzhan

Insert surname:
Zhazbulganov

Insert group:
IT02

```
PRESS 1 TO ADD STUDENT
PRESS 2 TO LIST STUDENTS
PRESS 0 EXIT
```

2

1 - Ilyas Zhuanyshev - IT01
2 - Erzhan Zhazbulganov - IT02

```
PRESS 1 TO ADD STUDENT
PRESS 2 TO LIST STUDENTS
PRESS 0 EXIT
```

0

Good Bye!

Awesome! Now, you can write small program.