6. Aplicaţii folosind operatori şi expresii C/C++ -Applications using C/C+ + operators and expressions

1. Obiective:

- Înțelegerea noțiunilor de expresie, operanzi și operatori
- Trecerea în revistă a principalilor operatori şi reținerea celor mai importanți
- Înțelegerea modului în care se face evaluarea expresiilor
- Scrierea și rularea de programe simple care folosesc operatori

1'. Objectives:

- Understanding the concept of expression, operand and operator
- Reviewing the main operators and remembering the most important ones
- Understanding the expression evaluation
- Writing and running some simple applications that use operators

2. Breviar teoretic

Operatorii sunt simboluri ce specifică operaţiile de efectuat asupra operanzilor. În urma aplicării unui operator se obţine un rezultat.

Operatorii limbajului C se pot grupa pe clase astfel:

Operatori aritmetici

unari : + (fără efect), - (negativare)

binari :

multiplicativi

*	înmulţire ;
/	împărţire, dacă operanzii sunt întregi rezultatul va fi partea întreagă a câtului;
%	modulo, doar pentru operanzi întregi şi are ca rezultat restul împărţirii întregi;

aditivi

+	adunare
−	scădere

Operatori de relaţie şi de egalitate

operatori de relaţie, care sunt: <, <=, >, >=,

operatori de egalitate, care sunt:

== egal

!= diferit.

Operatori logici

Operatori pe operand

! (**not**) negaţia logică, care se utilizează astfel:

!operand = 0 (False), dacă operand != 0

!operand = 1 (True), dacă operand = 0.

&& (**and**) şi logic,

E1 && E2, şi are valoarea 1 (True) dacă ambele expresii E1 şi E2 sunt True (!=0), în rest are valoarea False (0).

|| (**or**) sau logic,

E1 || E2 care are valoarea 0 (False) dacă ambele expresii E1 şi E2 au valoarea 0 (False). În rest expresia are valoarea 1 (True).

Operatori pe biţi

~ (**compl**) complement faţă de unu, schimbă fiecare bit 1 al operandului în 0, respectiv fiecare bit 0 în 1;

<< deplasare la stânga a valorii primului operand cu un număr de poziţii egal cu valoarea celui de-al doilea operand, forma de utilizare fiind: `op1 << op2`; este echivalent cu înmulţirea cu puteri ale lui 2;

>> deplasare la dreapta a valorii primului operand cu un număr de poziţii egal cu valoarea celui de-al doilea operand, forma de utilizare fiind: `op1 >> op2;` este echivalent cu împărţirea cu puteri ale lui 2;

& (**bitand**) şi logic pe biţi, se execută bit cu bit conform lui ŞI-LOGIC; este folosit la lucrul cu măşti pentru a selecta unii biti;

^ (**xor**) sau exclusiv pe biţi, se execută bit cu bit conform lui SAU-EXCLUSIV; este folosit pentru a anula sau seta diferiţi biţi.

| (**bitor**) sau logic pe biţi, se execută bit cu bit conform lui SAU-LOGIC; este folosit pentru a seta diferiţi biţi.

Operatori de atribuire (asignare)

asignare simplă, care se notează prin caracterul = şi are forma: v=(expresie) unde v este o variabilă simplă, referinţă la un element de tablou sau de structură. Se asociază de la dreapta la stânga astfel: v_n=…=v_1=v=expresie; asignarea făcându-se mai întâi v=expresie, apoi se asociază rezultatul lui v_1, apoi v_2 şi tot aşa până la v_n.

asignare compusă, caz în care pentru atribuire se foloseşte forma: operator=, unde operator poate fi unul binar aritmetic sau unul logic pe biţi, deci poate fi unul dintre operatorii următori: /, %, *, -, +, <<, >>, &, ^, | .

Sintaxa: var operator= expresie;

este echivalentă cu: var = var operator (expresie);

Remarcă: În versiunile inițiale, asignarea compusă era permisă și postfix. Când compilatorul acceptă atât =- cât și -=, unele compilatoare au adăugat un mesaj de avertizare pentru vechiul operator, ceva

de genul: „Operatorul =- este depreciat. Utilizați -=”, alte compilatoare ignoră operatorul postfix compus.

Operatori de incrementare şi decrementare (operatori unari). Aceştia sunt ++ şi --, putând fi postfixaţi (după operand), respectiv prefixaţi (înaintea operandului).

Operator de forţare a tipului sau de conversie explicită (cast)

`(tip) operand`, prin care valoarea operandului se converteşte la tipul indicat în C sau

`tip (operand)`, în C++.

Operator de determinare a dimensiunii

`sizeof(data)` sau `sizeof(tip)`, determinând dimensiunea în octeţi a unei date sau a unui tip;

Operatori de adresare (&, referenţiere) şi de indirectare (*, dereferenţiere)

Operatori paranteză

parantezele rotunde, ( )

paranteze pătrate, [ ]

Operator condiţional

`E1 ? E2 : E3`, unde E1, E2, E3 sunt expresii.

Operator virgulă

leagă două expresii în una singură astfel: `E1, E2`

Alţi operatori (pt. structuri)

punct .

săgeată ->

C++ defineşte cuvintele cheie `and, or, not, xor, and_eq, or_eq, not_eq, xor_eq`, etc. ca alternativa pentru `&&, ||, !, ^, &=, |=, != and ^=`, numite, **“digraphs”,** care pot fi folosite în C++.

(https://en.cppreference.com/w/cpp/language/operator_alternative)

Pentru compilatoarele C e necesar a include **<iso646.h>**

O expresie poate folosi operatori din clase diferite, la evaluarea ei ţinându-se cont de:

- precedenţă (prioritate): dă ordinea de efectuare a operaţiilor;
- asociativitate: indică ordinea de efectuare a operaţiilor care au aceeaşi precedenţă; poate fi stânga->dreapta (S->D) sau dreapta->stânga (D->S);
- regula conversiilor implicite: asigură stabilirea unui tip comun pentru ambii operanzi, la fiecare operaţie care solicită acest lucru şi în care tipurile diferă; regula de bază: se promovează tipul cu domeniu de valori mai mic către tipul cu domeniul de valori mai mare;

Ordinea de evaluare dată de precedență şi asociativitate poate fi modificată grupând operaţiile cu ajutorul parantezelor.

2. Theoretical breviary

Operators are symbols that specify the operations to be performed on operands. Following the application of an operator, a result is obtained.

C language operators can be grouped into classes as follows:

Arithmetic operators

unary: + (no effect), - (negation)

binary:

multiplication

* multiplication ;

/ division, if the operands are integer, the integer quotient will be returned;

% modulo, only for integer operands, obtains the remainder of the integer division;

addition

+ addition

- subtraction

Relational and equality operators

relational operators: <, <=, >, >=,

equality operators:

== equal

!= different

Logical operators

Logical operand operators

! (**not**) logical negation, which is used as it follows:

!operand = 0 (False), if operand != 0

!operand = 1 (True), if operand = 0.

&& (**and**) logical and,

E1 && E2 is evaluated to 1 (True) if both expressions E1 are E2 are True (!=0), otherwise False (0).

|| (**or**) logical or,

E1 || E2 is evaluated to False (0) if both expressions E1 and E2 are 0 (False). otherwise 1 (True).

Bitwise operators

~ (**compl**) complement, changes every bit of the operand from 1 to 0 and from 0 to 1;

<< (left shift), shifts to the left the value of the first operand by a number of positions equal to the value of the second operand, being used as: `op1 << op2;` is equivalent to multiplying by powers of 2;

>> (right shift), shifts to the right shift the value of the first operand by a number of positions equal to the value of the second operand, being used as: `op1 >> op2;` is equivalent to dividing by powers of 2;

& (**bitand**), bitwise AND applies a logical AND upon the corresponding bits of the operands; it is frequently used when working with masks selecting some bits;

^ (**xor**), exclusive OR, applies a logical XOR upon the corresponding bits of the operands; is used to cancel or set various bits.

| (**bitor**), bitwise OR, applies a logical OR upon the corresponding bits of the operands; is used to set various bits.

Assignment operators

simple assignment, which is denoted by the character = and has the form: v=(expression) where v is a simple variable, reference to an array or structure element. It is associated from right to left: v_n=…=v_1=v=expression; assigning first v=expression, the results are associated to v_1, then v_2, etc. until v_n.

compound assignment, in which a distinct operator is associated with the assignment operator (operator=), where the operator can be binary arithmetic or binary bitwise, so it can be one of the following operators: /, %, *, -, +, <<, >>, &, ^, | .

The syntax: var operator= expression;

is equivalent to: var = var operator (expression);

Remark: Early C versions allowed postfix compound assignment. When the compiler would accept both =- and -=, some compilers added a warning message for the old operator, something like: “Operator =- is deprecated. Use -=”, others ignore the compound postfix operator.

Increment and decrement (unary) **operators**. These are ++ and --, can be postfixed (after the operand), respectively prefixed (before the operand).

Type forcing operator (explicit conversion, cast)

`(type)operand`, by which the value of the operand is converted in C to the specified type, or `type(operand)` in C++.

Size determining operator

`sizeof(data)` or `sizeof(type)`, determining the size in bytes of a variable or type;

Addressing (&, referencing) and indirection (*, dereferencing) operators

Parenthesis operators

round parenthesis, ( )

square parenthesis, [ ]

Conditional operator

`E1 ? E2 : E3`, where E1, E2, E3 are expressions

Comma operator

links two expressions into a single one, like this: `E1,E2`

Other operators (used for structures)

point .

arrow ->

C++ defines the keywords `and, or, not, xor, and_eq, or_eq, not_eq, xor_eq`, etc. as an alternative for `&&, ||, !, ^, &=, |=, != and ^=`, named, **"digraphs"** which can be used in C++.

(https://en.cppreference.com/w/cpp/language/operator_alternative)

For C compilers is necessary to include **<iso646.h>**

An expression may use operators from different classes, being evaluated considering:

- the precedence (priority): reflects the order of performing operations;
- associativity: indicates the order in which operations having the same precedence are performed; can be left->right or right->left;
- Implicit conversions rule: ensures that a common type is established for both operands, each time the situation imposes it; Basic rule: the type with the smaller value range is promoted to the type with the larger value range;

The order of evaluation given by precedence and associativity can be modified by grouping the operations with round parentheses.

3. Exemple / Examples

Ex. 1

```
/* conditional operator */
#include <iostream>
using namespace std;

int main ( ){
       double n1, n2, n3;
       cout << "Enter 3 real numbers: ";
       cin  >> n1 >> n2 >> n3;
       cout << (n1 <= n2 && n2 <= n3 ? "Are ordered" : "Are not ordered") << '\n';
       return 0;
}//end_main
```

Ex. 2

```
/* logical and relational operators
The program checks if the character entered is an uppercase or lowercase letter
*/
#include <iostream>
using namespace std;

int isAlpha (char ch);

int main( ){
       char ch;
       cout << "Enter a character: ";
       cin >> ch;
       cout << (isAlpha(ch) != 0 ? "Is a letter" : "Is not a letter") << '\n';
       //cout << (isAlpha(ch) ? "Is a letter" : "Is not a letter") << '\n';
return 0;
}//end_main

int isAlpha (char ch){
       return ch >= 'a' && ch <= 'z' || ch >= 'A' && ch <= 'Z';
}// end_isAlpha
```

Ex. 3

```
/* a) bitwise operators - cout variant
The program applies the bitwise operators and displays the results in octal
*/
#include <iostream>
using namespace std;

int main( ){
unsigned char x = '\011';
unsigned char y = '\027';
cout << "x = " << oct << short(x) << '\n';
cout << "y = " << oct << short(y) << '\n';
cout << "~x = " << oct << (short)(~x) << '\n';
cout << "x & y = " << oct << (short)(x & y) << '\n';
cout << "x | y = " << oct << (short)(x | y) << '\n';
cout << "x ^ y = " << oct << (short)(x ^ y) << '\n';
cout << "x << 2 = " << oct << (short)(x << 2) << '\n';
cout << "x >> 2 = " << oct << (short)(x >> 2) << '\n';
return 0;
}//end_main
```

```
/* b) bitwise operators - printf variant
The program applies the bitwise operators and displays the results in octal
*/
#include <stdio.h>

int main() {
	unsigned char x = '\011';
	unsigned char y = '\027';
	printf("x = %o \n" , short(x));
	printf("y = %o \n", short(y));
	printf("~x = %o \n", short(~x));

	printf("x & y = %o \n", (short)(x & y));
	printf("x | y = %o \n", (short)(x | y));
	printf("x ^ y = %o \n", (short)(x ^ y));
	printf("x << 2 = %o \n", (short)(x << 2));
	printf("x >>2 = %o \n", (short)(x >>2));

	return 0;
}//end_main
```

Ex. 4

```
/* logical operators in literal representation */

#include <iostream>
#include <bitset>// display bitwise representation from STL
using namespace std;

int main( ) {
int a = 5, b = 9;  // a = 5(00000101), b = 9(00001001)
cout << "a = " << bitset<8> (a) << "," << " b = " << bitset<8>(b) << ", ! a = " <<
bitset<8>(!a) << "," << " not b = " << bitset<8>(not b) << endl;
```

```
            // The logical and bit result is 00000001
cout << "a & b = " << bitset<8>(a & b) << hex << " Hex value bitand: 0X" << (a
bitand b) << ", a && b = " << bitset<8>(a && b) << ", a and b = " << bitset<8>(a
and b) << endl;
            // The logical or bit result is 00001101
cout << "a | b = " << dec << bitset<8>(a | b) << hex << " Hex value bitor: 0X" <<
(a bitor b) << ", a || b = " << bitset<8>(a || b) << ", a or b = " << bitset<8>(a
or b) << endl;
            // The logical xor bit result is 00001100
cout << "a ^ b = " << dec << bitset<8>(a ^ b) << ", a xor b = " << bitset<8>(a xor
b) << hex << " Hex value: 0X" << (a xor b) << endl;
            // The logical complement bit result is 11111010
cout << "~(a=" << a << ") = " << dec << bitset<8>(~a) << hex << " Hex value compl:
0X" << (compl a) << endl;
            // The result is 00010010 for left shift
cout << "b << 1" << " = " << dec << bitset<8>(b << 1) << hex << " Hex value: 0X" <<
(b << 1) << endl;
            // The result is 00000100 for right shift
cout << "b >> 1 " << "= " << dec << bitset<8>(b >> 1) << hex << " Hex value: 0X" <<
(b >> 1) << endl;
}//end_main
```

Ex. 5

```
/* compound assignment operators
program that reads an integer from the console and performs with it:
multiplication, division modulo, bitwise XOR, bitwise AND
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main( ){
    int x;
    printf("Enter the value of int x: ");
    scanf("%d", &x);
    x ^= 0x0f;
    printf("\n%x", x);
    x %= 2;
    printf("\n%d", x);
    x &= 0x01;
    printf("\n%x\n", x);
    x *= 4;
    printf("\n%d", x);
    return 0;
}//end_main
```

Ex. 6

```
/* Program that reads an integer and applies the postfixed increment operator and
the prefixed increment operator, displaying the result each time
*/
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;

int main( ){
    int x;
    printf("Enter the value of x: ");
    scanf("%d", &x);

    printf("\n%d", x++);
    printf("\n%d", x);

    printf("\n%d", --x);
    printf("\n%d\n", x);

    //increment/decrement
    int a = 3;
    cout << "\n a initialized = " << a << '\n';//3
    int rez = a++ + ++a * a++;
    cout << "\n a processed = " << a << " rez= " << rez <<'\n'; //6, 20 or 28 or
    //33- depending on the compiler

    return 0;
}//end_main
```

Ex. 7

```
/* sizeof and casting operators
Program that displays the size (in bytes) for some data types and constants
*/


#include <iostream>
using namespace std;



int main( ){
       cout << "Size of char = " << sizeof(char) << " bytes\n";
       cout << " Size of char* = " << sizeof(char*) << " bytes\n";
       cout << " Size of short = " << sizeof(short) << " bytes\n";
       cout << " Size of int = " << sizeof(int) << " bytes\n";
       cout << " Size of long = " << sizeof(long) << " bytes\n";
       cout << " Size of float = " << sizeof(float) << " bytes\n";
       cout << " Size of double = " << sizeof(double) << " bytes\n";
       cout << " Size of 1.55 (double) = " << sizeof(1.55) << " bytes\n";
       cout << " Size of 1.55 (float) = " << sizeof((float)1.55) << " bytes\n";
       cout << " Size of 1.55L = " << sizeof(1.55L) << " bytes\n"; //8 or 16
       cout << " Size of SALUT = " << sizeof("SALUT") << " bytes\n";
       return 0;
}//end_main
```

4. Lucru individual:

1. Să se scrie un program care afişează valoarea polinomului de gradul 3 pentru o anumită valoare a variabilei reale x. Citiți coeficienții polinomului și valoarea x.
2. Să se scrie un program care citeşte lungimile laturilor unui triunghi (folosind variabile întregi). Validează că aceste 3 valori să fie laturi de triunghi şi afişează în acest caz laturile și aria triunghiului ca valoare reală într-un câmp de lățime 10, aliniat la stânga cu o precizie de 2 zecimale, altfel afişează cele 3 valori și un mesaj adecvat.
3. Să se scrie un program care afişează valorile biţilor unei variabile de tip unsigned char aplicând succesiv operatorul de deplasare dreapta şi operatorul %.
4. Să se scrie un program care monitorizează un canal de 16/32/64 biţi. Pentru aceasta citiţi de la tastatură o valoare întreagă fără semn x, care va fi afişată în zecimal, binar, octal şi hexazecimal. Folosiţi o funcţie pentru conversia numerelor din baza 10 în baza 2 sau facilitatea *bitset* din C++. Implementaţi o funcţie numită getsets() care primeşte trei valori ca parametri:

   x: valoarea citită de la tastatură

   p: poziţia unui bit din cei 16/32 sau 64 de biţi (numărând de la dreapta şi pornind de la 0)

   n: numărul de biţi care vor fi extraşi din valoarea citită.

   Funcţia returnează, aliniaţi spre dreapta, acei n biţi ai valorii x, pornind de la poziţia p, unde p<8*sizeof(x) şi p>n. Afişaţi rezultatul în binar, octal şi hexazecimal.
5. Să se scrie un program care citeşte de la intrarea standard un număr întreg şi afişează numărul de zerouri semnificative din reprezentarea sa binară.
6. Se citește de la intrarea standard o valoare întreagă. Să se afișeze în format zecimal valoarea fiecărui octet al întregului citit.
7. Se citesc de la tastatură 2 numere reale. Să se realizeze operațiile de adunare, scădere, înmulțire și împărțire cu valorile date. Să se afișeze rezultatele obținute, apoi să se rotunjească valorile obținute la valori întregi, folosind operatorul cast și fără a folosi funcții specifice. Să se afișeze apoi valoarea minimă dintre numerele citite folosind operatorul condițional.
8. Citiți de la tastatură mai multe caractere reprezentând litere mici. Să se transforme caracterele citite în litere mari în 2 moduri: printr-o operație aritmetică, respectiv folosind o operație logică și o mască adecvată.
9. Citiți de la tastatură 2 numere întregi. Dacă sunt egale, determinați aria cercului cu raza egală cu valoarea citită. Folosiți M_PI pentru calculul ariei din <cmath>. Dacă sunt diferite, calculați aria dreptunghiului cu laturile egale cu valorile date. Afișați aria calculată aliniată la stânga cu două zecimale precizie și valorile introduse într-un câmp de 10 poziții, specificând forma geometrică pentru care s-a făcut calculul.

4’. Individual work:

1. Write a program that displays the value of a 3-rd degree polynom, calculated in a certain point x. Read the coefficients of the polynom and the value of x from the KB.
2. Write a program that reads the lengths of the sides of a triangle (using integer variables). Validates that these 3 values are sides of the triangle and in this case displays the sides and area of the triangle as real value in a field of width 10, aligned to the left with a precision of 2 decimal values, otherwise it displays the 3 int introduced values and an appropriate message.

3. Write a program that reads an unsigned char value and displays the binary values resulting by successively applying the right shift operator and the % operator.
4. Write a program that monitors a communications channel on 16/32/64 bits. In order to do that, read from the keyboard an unsigned int value x, that will be displayed in decimal, binary, octal and hexadecimal counting bases. Use a function for converting the number from base 10 in base 2, or bitset facility provided in C++.

   Implement a function called getsets() that receives 3 parameters:

   x: the value read from the keyboard

   p: the position of a bit of those 16/32 or 64 bits (counting from the right, starting with 0).

   n: the number of bits that will be extracted from the number.

   The function returns, adjusted to the right, those n bits from the value x, starting with the position p, where p<8*sizeof(x) and p>n. Display the result in binary, octal and hexadecimal.
5. Write a program that reads from the standard input an integer number and displays the number of zeroes (significant) from it's binary representation.
6. Read an integer value from the standard input. Write, in decimal format, the value of each byte of the value.
7. Read 2 float numbers from the keyboard. Calculate the results obtained by applying the main arithmetic operations (+, -, *, /) upon them. Display the obtained results, then round the obtained values to integer values, using the cast operator and without using specific functions. Then display the minimum value of the numbers read using the conditional operator.
8. Read from the keyboard several lowercase characters. Convert the characters read to uppercase in 2 ways: by an arithmetic operation, respectively by using a logical operation and a suitable mask.
9. Read 2 integer values from the keyboard. If they are equal, determine the area of a circle that has the radius equal with the read value. Use M_PI from <cmath>. If the values are not equal, determine the area of the rectangle that has as sides the read values. In both cases, display the results aligned to the left with two decimal digits precision in a field of 10 digits width, and the values entered from the keyboard, also the name of the geometrical shape that corresponds to the calculated area.