7. Aplicaţii folosind instrucţiuni în C/C++. -Applications using instructions in C/C++

1. Obiective:

- Înţelegerea categoriilor de instrucţiuni din C/C++
- Înţelegerea modului în care lucrează instrucţiunile în C/C++
- Scrierea şi rularea de programe simple în care sunt folosite aceste instrucţiuni

1'. Objectives:

- Understanding the categories of C/C++ instructions
- Understanding the C/C++ instructions' operating mode
- Writing and running some simple programs that use instructions

2. Breviar teoretic

Se pot defini următoarele categorii de instrucţiuni.

Instrucţiunea compusă (secvenţială) este o secvenţă de [declaraţii şi] instrucţiuni, scrisă între acolade {....}

```
{
      //declaraţii
      //instrucţiuni
}
```

Instrucţiunea condiţională (de ramificaţie, decizională, alternativă)

```
if (condiţie) {
      //instrucţiuni
}
else {
      //alte instrucţiuni
}
```

Instrucţiuni ciclice (repetitive)

Instrucţiunea `while` (instrucțiune ciclică codiţionată anterior)

```
while (condiţie){ // antet
      //instrucţiuni (corpul buclei)
}
```

unde:

- dacă condiție != 0 (adevărat) atunci se execută secvența de instrucţiuni, după care se revine la punctul în care se evaluează din nou expresia, corpul executându-se atâta timp cât condiția este adevarată (condiție != 0); când condiție = 0 se trece la următoarea instrucţiune după ciclul while;
- dacă condiție = 0 de la început, atunci corpul nu se execută nicio dată.

Instrucţiunea `for` (instrucțiune ciclică codiţionată anterior):

```
for(exp1;exp2;exp3){
      //instrucţiuni (corpul buclei)
}
```

unde `exp1, exp2, exp3` sunt expresii cu următoarea semnificaţie:

- `exp1` reprezintă partea de iniţializare a ciclului for
- `exp2` reprezintă condiţia de continuare a ciclului for (are acelaşi rol cu condiția din ciclul while)
- `exp3` reprezintă partea de reiniţializare a ciclului for

În C++0x/1y/2z a fost introdusă o instrucţiune for pentru parcurgerea colecţiilor cu iteratori (for range based). Se utilizează pentru prelucrarea tuturor elementelor dintr-un domeniu.

```
for (declaratie : expresie_domeniu) {
      //instrucţiuni (corpul buclei)
}
```

unde:

- `declaratie` e declaraţia unei variabile de tipul unui element din domeniu sau o referinţă de acel tip. Deseori utilizează specificatorul `auto` pentru deducerea automată a tipului.
- `expresie_domeniu` sunt secvenţe de elemente cum sunt tablourile, containerele şi alte tipuri care definesc domeniul de valori.

Biblioteca STL conţine o funcţie template for_each( ) care lucrează asemănător cu instrucțiunea for range based.

Instrucţiunea do-while (instrucțiune ciclică condiţionată posterior):

```
do{
      //instrucţiuni, se executa cel putin o data
}
while(condiție);
```

Instrucţiunea selectivă

```
switch (expresie){
      case c1:    instructiuni_1
            [break;]
      case c2:    instructiuni_2
            [break;]
      ...
      case cn:    instructiuni_n
                  [break;]
      [default:   alte_instructiuni]
}
```

unde:

- `c1, …, cn` sunt constante ce pot fi: întregi, caractere, elemente de enumerare.
- `instrucţiuni_1, …, instrucţiuni_n, alte_instrucţiuni`, sunt secvenţe de instrucţiuni.

Noile compilatoare C++ (nu de la Microsoft) permit instrucţiuni `switch-range`, cu valori într-un domeniu pentru `ci`.

Funcţii şi instrucţiuni de salt

Funcţia `exit()`, declarată în `<stdlib.h>`, are următorul prototip:

```
void exit(int cod); // termină programul în curs de execuţie
```

Instrucţiunea `continue`, are următorul prototip:

```
continue;   //termină iteraţia curentă
```

Instrucţiunea `break`, are următorul format:

```
break;  //termină bucla curentă
//sare la prima instructiune după linia finală a buclei
```

Instrucţiunea `return`, are următoarele formate:

```
return;    //revine din funcţia curentă, fără o valoare asignată
return expresie; /*revine din funcţia curentă, returnând
valoarea expresiei */
```

Instrucţiunea `goto`, are următorul format:

```
goto eticheta;      /* eticheta este un identificator al unei alte
      secvente de cod */
```

2’. Theoretical breviary

The following categories of instructions can be defined.

The compound (sequential) instruction is a sequence of [statements and] instructions, written between curly braces `{....}`

```
{
      //declarations
      //instructions
}
```

The conditional instruction (logical branches, decision-making, alternative)

```
if (condition) {
      //instructions
}
else {
      //other instructions
}
```

Loop (repetitive) instructions

`while` instruction (pre-test loop)

```
while (condition){      //header
      //instructions (loop body)
}
```

where:

- if condition != 0 (true) the sequence of instructions is executed, after which the program returns to the point where the expression is re-evaluated, the loop body being executed as long as condition is true (condition != 0); when condition = 0 (false) the program proceeds to the next instruction after the while cycle;
- if condition = 0 from the beginning, the loop is never executed.

`for` instruction (pre-test loop)

```
for(exp1;exp2;exp3){
      //instructions (loop body)
}
```

where `exp1, exp2, exp3` are expressions with the following meaning:

- `exp1` represents the initialization part of the loop
- `exp2` represents the loop continuing condition (has the same role as the condition in the while cycle)
- `exp3` represents the re-initialization part of the for loop

In C++0x/1y/2z a specific for instruction has been introduced for iterating collections (for range based). It is used to process all elements in a domain.

```
for (declaration: domain_expression) {
      //instructions (loop body)
}
```

where:

- `declaration` is declaration of a variable having the type corresponding to the elements in the domain. Frequently the `auto` specifier is used for automatically determining the variable's type.
- `domain_expression`: sequences of items such as arrays, containers, and other types that define the collection.

The STL library contains a template function named `for_each( ) that wors as for range based.`

do-while instruction (posterior loop instruction)

```
do{
      //instructions, executed at least once
}
while(condition);
```

The selective instruction

```
switch (expression){
      case c1:    //instructions_1
                  [break;]
      case c2:    // instructions_2
                  [break;]
      ...
      case cn:    // instructions_n
                  [break;]
      [default:   //other_instructions]
}
```

where:

- `c1, …, cn` are constants having as type: integer, character, enumeration elements.
- `instructions_1, …, instructions_n, other_instructions`, are sequences of instructions.

New compilers (not from Microsoft) allow switch range, as a range value for `ci`.

Jump functions and instructions

Function `exit()`, declared in `<stdlib.h>`, has the following prototype:

```
void exit(int cod); // terminates the program
```

Instruction `continue`, has the following prototype:

```
continue;   //terminates the current iteration
```

Instruction `break`, has the following format:

```
break;
//terminates the current loop
//jumps to the first instruction after the final line of the loop
```

Instruction `return`, has the following formats:

```
return;    /*returns from current function without having a value
assigned */
```

```
return expression; /*returns from the current function, returning the
expression value */
```

Instruction `goto`, has the following format:

```
      goto label;      // label is an identifier of another code sequence
```

3. Exemple / Examples

Ex. 1

```
/* Program that reads multiple integers in a one-dimensional array, validating the
number of elements, and then display them */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define MAX  10

int main() {
    int i, n, tab[MAX];
    //reading the number of values
    do
    {
        printf("\nEnter the number of values n: ");
        scanf("%d", &n);
    if (n <= 0 || n > MAX)printf("\n\t Incorrect size (must be >0 and <=%d !", MAX);
    } while (n <= 0 || n > MAX);

    printf("\nEnter %d integer numbers: ", n);
    for (i = 0; i < n; i++) {
        printf("\n\t Waiting for number %d : ", i);
        scanf("%d", &tab[i]);
    } // end for
    //... data processing
    printf("\nYou entered the following values :\n");
    for (i = 0; i < n; i++)
        printf("\t%d\n", tab[i]);
    return 0;
}//end_main
```

Ex. 2

```
/* program that reads a number in the range [1...7] and displays the name of the
corresponding day of the week */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int  main( ){
       int i;
       puts("Enter an int number in the interval [1,7]: ");
       scanf("%d", &i);

       switch(i){
       case 1:
              puts("Monday");
              break;
       case 2:
              puts("Tuesday");
              break;
```

```
	case 3:
		puts("Wednesday");
		break;
	case 4:
		puts("Thursday");
		break;
	case 5:
		puts("Friday");
		break;
	case 6:
		puts("Saturday");
		break;
	case 7:
		puts("Sunday");
		break;
	default:
		puts("Wrong number");
	}
	return 0;
}//end_main
```

Ex. 3

```
/* Adds the values entered from the keyboard. Confirmation required */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
//#include <conio.h>

int main( ){
	int m, sum = 0;
	char key;
	do{
		printf("Enter a number: ");
		scanf("%d", &m);
		sum += m;
		printf("Press any key to continue. N or n stops the reading: \n");
		//key = _getch();
		scanf(" %c", &key);
	} while (!((key == 'n') || (key == 'N')));
	printf("\nThe sum of the entered numbers is: %d ", sum);
	return 0;
}//end_main
```

Ex. 4

```
/* The program evaluates whether an entered number is prime or not. Optimize the
algorithm */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int nrPrim(int);//prototype

int main( ){
	int nr;
	printf("\nEnter a number : ");
	scanf("%d", &nr);
	if(nrPrim(nr))
		printf("\nNumber %d is prime\n", nr);
```

```
	else
		printf("\nNumber %d is not prime\n", nr);
	return 0;
}// end_main

// The function returns 1 if n is prime or 0 if it is not
int nrPrim(int n){
	int i;
	if(n <= 1)
		return 0;
	if(n == 2)
		return 1;
	if(n % 2 ==0)
		return 0;
	for(i=3; i<n; i+=2) {
		if(n % i == 0)
			return 0;
	} // for
	return 1;
}// end_nrPrim
```

Ex. 5

```
/* Program that displays the values in a one-dimensional array, using 2 variants of
the for loop */
//#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;

int main( ) {
	int arr[ ] = { -1, 20, 3, 4 }; // pre-initialized one-dimensional array
	cout<< "The array has: " << _countof(arr) <<" elements.\nThey are: \n";

	for (int i : arr)  // iterator with specified type
		cout << i << " ";

	cout << "\nThe values will be modified!\n";
	for (auto& i : arr) { // iterator with auto reference
		cout << "\nThe value of the current item is = " << i << "\t enter a
new value: ";
		cin >> i;// scanf("%d", &i); //read inside the array
	// both work the same, scanf() requires #define _CRT_SECURE_NO_WARNINGS
	}

	cout << "\nThe new array (using sizeof):\n";
	//for (int i = 0;i < sizeof(arr)/sizeof(int); i++)//inefficient standard for
	constexpr int dim = sizeof(arr)/sizeof(int);
	for (int i = 0;i < dim; i++)// standard for
		cout << arr[i] << ' ';

	cout << "\n\nThe new array, (using _countof)" << '\n';
	for (int i = 0; i < _countof(arr); i++)//inefficient standard for
		cout << arr[i] << ' ';
	cout << '\n';

	cout << "\nfor-range access to an init list\n";
	for (int i : {-1, 1, 3, 5, 7}) // the initializer may be a braced-init-list
```

```
            cout << i << ' ';
      cout << '\n';

      cout << " \nAnother display...for-range as counter\n";
      for (int i : arr)
            cout << 7 << ' ';

      cout << "\nArray of pointers to constant character strings\n";
      const char* words[ ] = { "alfa", "beta", "gama" };
      for (const char* word : words)
            cout << word << '\t';

      return 0;
} // main
```

Ex. 6

```
/* Switch range gcc compilers, not VC++ Microsoft */
#include <iostream>
using namespace std;

int main( ){
      int score;
      cout << "Score values 0-100:";
      cin >> score;
      switch(score){
       case 0:
            cout << "a"; break;
       case 1 ... 10:
            cout << "b"; break;
       case 11 ... 24:
            cout << "c"; break;
       case 25 ... 49:
            cout << "d"; break;
       case 50 ... 100:
            cout << "e"; break;
        default:      cout << "BAD VALUE";
      }
      cout << endl;

}//end_main
```

Ex. 7

```
/* for with break and continue */

//a)The break statement is used to exit a loop early. Here's an example
with a for loop

#include <iostream>

constexpr int max =10;


int main() {

    for (int i = 0; i < max; ++i) {

        if (i == 4) {

            break; // Exit the loop when i is 4
```

```
        }
        std::cout << i << std::endl;
    }
    return 0;
}//main


//b)The continue statement skips the current iteration of the loop and
proceeds with the next iteration.
#include <iostream>
constexpr int max =10;

int main() {
    for (int i = 0; i < max; ++i) {
        if (i == 4) {
           continue; //Skip the value when i is 4,continue rest of the loop
        }
        std::cout << i << std::endl;
    }
    return 0;
}//end_main
```

4. Întrebări:

1. Evidenţiaţi diferenţa dintre structura *while* şi *do...while* printr-un exemplu.
2. Care este rolul instrucţiunii *break* în cadrul instrucţiunii *switch*?
3. Care este rolul instrucţiunii *break* în cadrul instrucţiunilor ciclice?
4. Care este rolul instrucţunii *continue* în cadrul instrucţiunilor ciclice ?

4'. Questions:

1. Give an example that highlights the difference between the while structure and the do... while.
2. What is the role of *break* instruction inside a *switch*?
3. What is the role of *break* instruction in cyclic instructions?
4. What is the role of *continue* instruction in cyclic instructions?

5. Teme:

1. Se citesc trei numere de la tastatură a, b şi c. Să se determine aria dreptunghiului ale cărui laturi sunt a şi b. Verificaţi dacă diagonala dreptunghiului este egală cu c, considerând o precizie eps=0.01 definită iniţial.
2. Să se scrie un program care verifică dacă un număr citit de la tastatură este pătrat perfect.
3. Să se scrie un program care calculează a^n, prin înmulţiri repetate ale lui a, unde n este citit de la tastatură (a se defineşte în program sau se citeşte de la tastatură).
4. Să se scrie un program care citeşte de la tastatură o valoare întreagă n şi calculează n! (n-factorial) în mod nerecursiv.

5. Să se scrie un program care citeşte de la intrarea standard un număr dat şi:
    - determină cel mai mare număr prim mai mic decât numărul dat
    - determină toate numerele prime mai mici decât numărul dat.
6. Să se scrie un program care determină cel mai mare divizor comun a doi întregi introduşi de la tastatură.
7. Să se scrie un program care determină toţi divizorii unui număr introdus de la tastatură.
8. Calculaţi produsul a două numere întregi folosind numărul corespunzător de adunări ale primului număr, dat de al doilea.
9. Să se scrie un program care determină câtul împărţirii a doi întregi introduşi de la tastatură folosind scăderi succesive.
10. Să se scrie un program care determină numărul de cifre care compun un număr întreg citit de la tastatură.
11. Să se scrie un program care citeşte de la tastatură n numere întregi. Afişează toate numerele impare introduse şi le stochează într-un tablou.
12. Să se citească un număr întreg r de la tastatură. Se citesc apoi numere reale, până când suma lor depăşeşte valoarea lui r. Să se afişeze suma numerelor citite, cu o precizie de 2 zecimale la partea fracţionară şi numărul lor (câte s-au introdus).
13. Să se scrie un program care determină cmmmc a două numere citite de la tastatură.
14. Scrieţi un program care citeşte n numere întregi de la tastatură şi le afişează pe cele divizibile cu 3.
15. Să se scrie un program care citeşte de la tastatură un caracter, pe care îl afişează pe n rânduri, câte n caractere pe un rând, n citit de la tastatură.
16. Să se scrie o aplicaţie C/C++ în care se introduc de la tastatură numere întregi, până ce utilizatorul apasă tasta <Esc>, sau o altă tastă predefinită. Să se determine şi să se afişeze media numerelor impare pozitive care au fost introduse.

5’. Homework:

1. Read from the keyboard 3 numbers a, b and c. Determine the area of the rectangle that has the sides equal to a and b. Verify if the rectangle’s diagonal is equal to c with a precision eps=0.01 initial established.
2. Please verify if a natural number introduced from the keyboard is a perfect square or not.
3. Write a program that calculates a^n, by repeated multiplications of a where n is read from the keyboard (a is “hard coded”, or introduced from the KB).
4. Write a program that reads from the keyboard an integer value n and calculates n! (n-factorial) not in a recursive mode.
5. Write a program that reads from the KB a given number and:
    - determines the greatest prime number that’s smaller than the given number.
    - determines all the prime numbers smaller than the given number.
6. Write a program that determines the greatest common divider of 2 integer values read from the keyboard.
7. Write a program that determines all the divisors of a value introduced from the keyboard.

8. Calculate the product of 2 integer numbers using additions of the first number given by the second one.
9. Write a program that determines the integer quotient of 2 integer numbers introduced from the KB using a series of subtractions.
10. Write a program that determines the number of figures that compose an integer number read from the keyboard.
11. Write a program that reads from the keyboard n integer numbers. Display all the odd numbers and store them in an array.
12. Read from the keyboard an integer number r. Read a series of real numbers, until their sum is greater than r. Display the sum with a 2 digits precision at the fractional part and how many numbers were introduced.
13. Determine the least common multiple of 2 integer numbers read from the keyboard.
14. Write a program that reads n integer numbers from the keyboard and displays the values that can be divided by 3.
15. Write a program that reads from the keyboard a single character. The program should display that character on n rows, n times in a row, n read from the keyboard.
16. Write a C/C++ application that reads from the keyboard a series of integer numbers, until the user presses the <Esc> key, or another predefined key. Determine and display the average value of the odd positive numbers.