RO: Aplicaţii bazate pe macrofuncţii, aserţiuni, funcţii inline, funcţii cu parametri impliciti, funcţii cu număr variabil de parametri, supraîncărcarea funcțiilor.

EN: Macro functions, assertions, inline functions, functions with implicit parameters, functions with a variable number of parameters, overloading functions.

Obiective:

- Înţelegerea noţiunilor legate de functii macro, funcţii inline prin aplicarea lor în practică în programe C/C++;
- Asimilarea modului de lucru cu funcţii cu parametri impliciţi şi cu parametri variabili.

Rezumat:

Funcţiile macro (macrodefiniţii cu parametri):

Au următoarea sintaxă: *#define nume(p1, p2,...,pn) text*

unde:

- p_i sunt parametri formali;
- *text* este textul de substituţie care va conţine parametri formali.

Apelul unei macrodefiniţii cu parametri se face analog apelului unei funcţii, dar prin expandare: numele macro-ului urmat de parametri efectivi între paranteze şi separaţi prin virgule.

Macrourile predefinite sunt*: __cplusplus, __LINE__, __DATE__ , __FILE__, __TIME__ , __STDC__.*

Anularea unei macrodefiniţii se face cu directiva *#undef.*

Aserţiunile, sunt utilizate pentru gestionarea simplă a excepțiilor si sunt introduse prin macrofunctia *assert(...).*
Macrofunctia *assert (...),* e definită în fișierul antet *assert.h*, testează valoarea unei expresii.
Dacă valoarea expresiei este 0 (fals), atunci *assert (...)* afisează un mesaj de eroare și apelează funcția *abort ()* (din - *stdlib.h*) pentru a termina executarea programului.
Când asertiunile nu mai sunt necesare, linia:
#define NDEBUG
este inserată în fișierul programului, mai degrabă decât ștergerea manuală a fiecărei asertiuni.
Din *C ++ 1y* putem folosi *static_assert* care este este aplicată în timpul compilării.

Funcţiile inline:

- sunt specifice limbajului C++;
- se definesc ca şi celelalte funcţii, în plus se adaugă cuvântul cheie "inline";
- păstrează proprietăţile funcţiilor legate de verificările de la apel (număr şi tip pentru parametri), de modul de transfer al parametrilor, de domeniul declaraţiilor locale şi al parametrilor;
- compilatorul substituie textul corespunzător funcţiei;
- se folosesc atunci când funcţiile ocupă un număr mic de linii de cod, pentru a creşte viteza de lucru, evitându-se astfel operaţiile specifice la apelul unei funcţii (încărcarea şi descărcarea stivei etc.).

Funcţiile cu parametri impliciţi:

- permit declararea de valori implicite pentru parametri ;
- la apel se poate omite specificarea parametrilor efectivi pentru acei parametri formali cu valori implicite, pentru care compilatorul transferă automat acele valori implicite;
- valorile implicite se specifică o singură dată în prototip sau funcţie;
- parametri cu valori implicite trebuie să apară de la sfârșitul listei.

Funcţiile cu un număr variabil de parametri:

- sunt necesare în situaţii similare lui *printf()* şi *scanf();*
- au prototipul de forma:
  tip_returnat nume_functie(tip arg1, ...);
- pot accesa argumentele din listă utilizând un set de funcţii din fişierul antet *stdarg.h*:
  void va_start(va_list ap, lastfix);
  type va_arg(va_list ap, type);
  void va_end(va_list ap);

Supraîncarcarea funcţiilor:

In C++ e posibil ca funcţii diferite să aibă acelaş nume dar să difere prin *semnătură.*

Semnătura e dată de: *numele funcţiei, numărul de parametri, tipul parametrilor, poziţia*; valoarea de retur nu e considerată.

Exemple:

```
//*****************************************************************************
//Macrofunctii
A)#define MAX2(a,b) ((a)>(b)?(a):(b))
```

```
Apel: cin >>a; cin >>b;
cout<< "\nMax (2 valori) = "<< MAX2(a,b);
B)#define MAX2I(a,b) {if(a<b)\
                                a=b;\
                                }
 ...
Apel: cin >>a; cin >>b;
MAX2I(a,b);
cout<< "\n Max (cu if) = "<< a;

//*****************************************************************************************
// inline_test
#include <iostream>
using namespace std;

inline int max(int a, int b);

int main()
{
        int v1, v2, max_v;
        cout << "Introduceti 2 valori intregi: ";
        cin >> v1;
        cin >> v2;
        max_v = max(v1, v2);
        cout << "\n Maximul este= " << max_v << endl;
        }

inline int max(int a, int b)
{
        if (a > b) return a;
        return b;
}

//*****************************************************************************************
//Asertiuni validare date
#define _CRT_SECURE_NO_WARNINGS
#include <assert.h>
#include <stdio.h>
#include <string.h>

int main( ) {
   int a;
   char str[50];
   printf("Enter an integer value: ");
   scanf("%d", &a);
   assert(a >= 5);
   printf("Integer entered is %d\n", a);
   printf("Enter string: ");
   scanf("%s", str);
   assert(strlen(str)>5);
   printf("String entered is: %s\n", str);
    return 0;
}
//*****************************************************************************************
//Invalidare asertiuni
#include <iostream>
// assert( ) disabled
#define NDEBUG
#include <cassert>
//static_assert(sizeof(int) == 4, "int must be 4 bytes");//compilation time
using namespace std;

int main( ){
```

```
   assert(2 + 2 == 3 + 1);
   cout << "Expression valid...Execution continues.\n";
   assert(2 + 2 == 1 + 1);
   cout << "Assert disabled...execution continuous with invalid expression\n";
}
//*****************************************************************************************
//Ilustreaza folosirea unei functii (itoa( )) cu un parametru implicit (aici radix) - deprecated
#define _CRT_SECURE_NO_WARNINGS
#define _CRT_NONSTDC_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#define DIM 10
//declaram functia itoa() ca avand un parametru implicit baza de numeratie = 10
char *itoa(int value, char *string, int radix=10);

int main( )
{
        int i;
        char sir_baza10[DIM] , sir_baza16[DIM];
        //apelam functia cu parametrul radix=10, adica se va transforma numarul intreg
        //in sirul de numere care ii corespunde in baza zece
        printf("Numarul intreg este: ");
        scanf("%d",&i);
        //apelul s-a facut fara a preciza al treilea parametru, care se considera 10
        itoa(i,sir_baza10);
        printf("Sirul de caractere corespunzator numarului intreg in baza 10 este: %s\n", sir_baza10);
        //apelul s-a facut cu al treilea argument al functiei itoa avand valoarea 16
        itoa(i,sir_baza16,16);
        printf("Sirul de caractere corespunzator numarului intreg in baza 16 este: %s", sir_baza16);
}

//*****************************************************************************************
//Ilustreaza folosirea unei functii (_itoa_s( )) fara param. impliciti din stdlib.h
#include <stdio.h>
#include <stdlib.h>

const int DIM = 10;
char* _itoa_s(int value, char* string, int length, int radix=10);//e acceptat ca param dar trebuie dat la apel

int main()
{
int i;
char sir_baza10[DIM], sir_baza16[DIM];
printf("Numarul intreg este: ");
scanf_s("%d", &i);
//apelul s-a facut cu al patrulea argument al functiei _itoa_s( ) avand valoarea 10
//_itoa_s(i, sir_baza10, _countof(sir_baza10));//linker error
_itoa_s(i, sir_baza10, _countof(sir_baza10), 10);
printf("Sirul de caractere corespunzator numarului intreg in baza 10 este: %s\n", sir_baza10);
//apelul s-a facut cu al patrulea argument al functiei _itoa_s( ) avand valoarea 16
_itoa_s(i, sir_baza16, _countof(sir_baza16), 16);
printf("Sirul de caractere corespunzator numarului intreg in baza 16 este: %s", sir_baza16);
}
//*****************************************************************************************

//polinom este o functie care calculeaza valoarea unui polinom, avand un prim parametru valoarea variabilei
//independente si apoi un numar variabil de parametri ce corespund coeficientilor polinomului
//functia polinom va determina gradul polinomului din numarul de parametri

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdarg.h>
#include <math.h>
```

```
double polinom(double, ...);

int main()
{
	double x;

	printf("Introduceti valoarea lui x: ");
	scanf("%lf", &x);
	printf("\nValoarea lui P(x) = 5*x^3+34*x^2+20*x-5 este: %lf",
		polinom(x, -5.0, 20.0, 34.0, 5.0, HUGE_VAL));
	// HUGE_VAL este o valoare particulara pentru care se opreste cautarea in lista de parametri,
	//are ca valoare cel mai mare numar real (double) reprezentabil
	printf("\n\n");
	printf("\nValoarea lui P(x) = x^5+3*x^4-x^2+2*x+15 este: %lf\n",
		polinom(x, 15.0, 2.0, -1.0, 0.0, 3.0, 1.0, HUGE_VAL));
}//end main

double polinom(double x, ...) //ordinea coeficientilor este de la gradul 0 la gradul n
{
	int grad_polinom = 0;		//gradul polinomului
	double rezultat = 0., coef;	 //coef  ia valoarea coeficientului la o anumita putere
	va_list ap;//declarare pointer
	va_start(ap, x);//definire pointer
	while ((coef = va_arg(ap, double)) != HUGE_VAL)
	{
		rezultat += coef * pow(x, grad_polinom);
		grad_polinom++;
	}//end while

	va_end(ap);

	printf("\nGradul polinomului este: %d\n", grad_polinom - 1);
	return rezultat;
}//end polinom
//*****************************************************************************
//supraincarcarea functiilor, apel prin valoare si prin referinta
#include <iostream>
using namespace std;

int abs(int n);//by value
int abs(int& n, int b);//by reference

int main() {
	int a = -8;
	cout << "\nAbs. by value =" << abs(a);
	cout << "\nAbs. by ref =" << abs(a, 2);
}

int abs(int n) {
	return ((n < 0) ? (-n) : n);
}

int abs(int& n, int b) {
	return ((n < 0) ? (-n) : n);
}
//*****************************************************************************
```

Teme:

1. Definiţi o funcţie macro MAX care determină şi afişează maximul dintre 2 si dintre 3 numere introduse de la tastatură.
2. Definiţi o funcţie *inline min( )* care determină şi afişează minimul dintre 2 si dintre 3 numere întregi introduse de la tastatură.

3. Considerati o structura de date *Student*, care contine un camp de tip sir de caractere (maxim 30) pentru *nume_prenume* si un alt camp *nota* de tip *int*. Definiti un obiect de tip *Student* la care datele vor fi citite de la tastatura. Validati ca *nume_prenume* sa aiba cel putin 5 caractere iar *nota* sa fie >=5 si <=10. Afisati campurile obiectului in caz de introducere corecta.
4. Consideraţi o funcţie cu 3 parametrii toţi impliciţi (int, float, double) care returnează produsul acestor valori. Apelaţi funcţia considerând mai multe variante de apel concrete (fără parametri, 1 parametru, 2 parametri, 3 parametri).
5. Folosind supraîncărcarea funcţiilor definiţi trei funcţii cu acelaşi nume dar cu tipuri diferite de parametri (int, int *, int&) care returnează radicalul unei valori intregi. Analizaţi cazul transmiterii parametrilor prin valoare şi prin referinţă.
6. Determinaţi minimul dintr-un şir de 10 numere flotante (introduse de la tastatură/iniţializate) folosind funcţii cu un număr variabil de parametri. Se vor considera primele 7 valori din şir, apoi următoarele 3, după care se afişează minimul din cele 10 folosind valorile determinate anterior.
7. Scrieţi un program care face o codare simplă prin adaugarea la codul ASCII al caracterului, o valoare n=3, folosind macro funcții. Exemplu: ‘a’ devine în urma codării ‘d’.
8. Să se scrie un program care afişează numele programului, data şi ora compilării şi numărul de linii pe care îl are acest program.
9. Realizaţi o aplicaţie C/C++ care aplică un cod binar (mască) fiecărui element al unui şir printr-o funcţie de codare şi invers îl decodează într-o funcţie de decodare, folosind funcţii macro. Exemplu: fie caracterul *‘a’*, codul mască 11001010:
    a. rezultatul codării ar fi:
    ‘a’ -> 97 ASCII ->01100001 SAU EXCLUSIV logic pe biţi
    codul binar 11001010

    10101011
    b. decodarea se face:
    rezultatul codării ->10101011 SAU EXCLUSIV logic pe biţi
    codul binar 11001010

    01100001 codul ASCII al lui *‘a’*

Homework:

1. Define a macro function MAX that determines and display the maximum among 2 and 3 numbers introduced from the KB.
2. Define an inline function *min()* that determines and display the minimum among 2 and 3 numbers introduced from the KB.
3. Consider a *Student* data structure, which contains a string field (maximum 30) for *name_surname* and another *note* field of type *int*. Define a *Student* object where the data will be read from the keyboard. Validate that *name_surname* has at least 5 characters and the *note* should be> = 5 and <=10. Display the object fields if entered correctly.
4. Consider a function with 3 implicit parameters (all) (int, float, double) that returns the product of the values. Call that function with different variants for effective parameters(no pram, 1 param, 2 params, 3 params).
5. Using functions overloading define 3 functions with the same name but with different prams type (int, int*, int&) that will return the square root of the int value. Analyze the calling mechanism by value and reference.
6. Determine the minimum of a 10 float numbers from a string (implicit values or from the KB) using a function with a variable number of parameters. The first 7 values will be considered initially, next the last 3, and at the end these 2 values.
7. Write a program that performs a simple coding operation by increasing with 3 the value of the ASCII code of a character using macro functions. For example, A becomes C, etc.
8. Write a program that displays the name of the program, the compilation date and time and the number of code lines included in the program.
9. Implement a C/C++ application that applies (using a macro function) a binary mask to each element located in an array of characters. Define the decoding function, too. Example: considering the character *a* and the mask code 11001010:
    a. the coding result will be obtained as it follows:
    ‘a’ -> 97 ASCII ->01100001 EXCLUSIVE OR
    Binary code 11001010

    10101011
    b. the decoding process:
    the coded result -> 10101011 EXCLUSIVE OR
    binary code 11001010

    01100001 ⇔the ASCII code of *a*