POO – C++ - Laborator 11

Cuprins

1. Biblioteca standard C++

Funcțiile bibliotecii standard din C++ pot fi împărțite în două categorii:

Biblioteca **funcțiilor standard** conține funcții generice de sine stătătoare care nu pot fi încadrate în nici o clasă. Este moștenită din C. Biblioteca de funcții standard este împărțită în următoarele categorii:

- Funcții de intrare/ ieșire (I/O);
- Funcții pentru șiruri de caractere și pentru manipularea acestora;
- Funcții matematice;
- Funcții pentru managementul timpului, a datei și a localizării;
- Funcții pentru alocare dinamică;
- Funcții auxiliare;
- Funcții support pentru diferite seturi de caractere.

Clasele orientate obiect sunt reprezentate de o colecție de clase și de funcțiile associate acestora. Biblioteca standard de clase orientate obiect din C++ definește o gamă variată de clase asigurând suportul pentru numeroase operații de bază, inclusive I/O, șiruri sau procesări numerice. Biblioteca cuprinde următoarele clase:

- Clase standard C++ de intrare/ieșire;
- Clasa String;
- Clase numerice;
- Algoritmi STL;
- Iteratori STL, etc.

1. Funcții standard de intrare/ieșire

Fișierul Header	Funcții și descriere
<iostream>	Acest fișier definește funcțiile **cin, cout, cerr** și **clog** corespunzătoare intrării standard, ieșirii standard, erorii standard fără memorie-tampon și erorii standard cu memorie-tampon.
<iomanip>	Acest fișier declară servicii utile pentru desfășurarea operațiilor de intrare/ieșire cu format prin intermediul manipulatorilor parametrizați cum ar fi **setw** și **setprecision**.
<fstream>	Acest fișier declară sevicii folosite pentru procesarea controlată a fișierelor.

```
#include <iostream>

using namespace std;

int main()
{
	//cout
	char str[] = "Hello C++";
	cout << "Valoarea sirului este : " << str << endl;

	//cin
	char name[50];
	cout << "Introduceti numele: ";
	cin >> name;
	cout << "Numele introdus este: " << name << endl;

	//cerr
	char stre1[] = "Citire nereusita....";
	cerr << "Mesaj eroare : " << stre1 << endl;

	//clog
	char stre2[] = "Citire nereusita....";
	clog << "Mesaj eroare : " << stre2 << endl;


	return 0;
}
```

Cele 3 streamuri de ieşire se folosesc, de obicei, pentru:

- std::cout – Regular output (console output)
- std::cerr – Error output (console error)
- std::clog – Log output (console log)

În cele mai multe cazuri aceste streamuri sunt direcţionate spre ieşirea standard (consola), dar ele pot fi re-direcţionate în funcţie de specificaţiile programului. De exemplu, `cout` poate fi direcționat pentru a scrie într-un fişier iar cerr pentru a scrie în alt fişier.

Aceste redirecţionări se fac cu funcţia din libraria standard `std::ios::rdbuf()`. Pentru un exemplu de folosire a acestei funcţii, consultaţi pagina http://www.cplusplus.com/reference/ios/ios/rdbuf/

Atât `cout` cât şi `cerr` şi `clog` sunt obiecte de tipul `ostream` (output stream). Această clasă permite setarea unor parametri în funcţie de care se va face scrierea propriuzisă în bufferul de ieşire. Pentru o listă completă a acestor membri, accesaţi http://www.cplusplus.com/reference/ostream/ostream/

2. Funcții standard pentru manipularea șirurilor de caractere

String-urile sunt obiecte reprezentate sub forma unor secvențe de caractere. Clasa string standard oferă suport pentru aceste obiecte prin implementarea unor metode specifice de prelucrare a stringurilor formate din caractere reprezentate pe un singur octet. C++ oferă două tipuri de reprezentări pentru șirurile de caractere:

- Reprezentarea tip C unui șir de caractere;

Indexul	0	1	2	3	4	5
Variabila	H	e	l	l	o	\0
Adresa	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
	char greeting1[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };
	char greeting2[] = "Hello";

	cout << "Greeting message1: ";
	cout << greeting1 << endl;

	cout << "Greeting message2: ";
	cout << greeting2 << endl;

	return 0;
}
```

C++ oferă o gamă variată de funcții pentru manipularea șirurilor care se termină cu '\0':

Funcția	Descriere
strcpy_s(s1, dim, s2);	Copie șirul s2 în s1.

strcat_s(s1, dim, s2);	Concatenează șirurile s1 și s2, prin adăugarea lui s2 la sfârșitul lui s1.
strlen(s1);	Returnează lungimea lui s1.
strcmp(s1, s2);	Returnează 0 dacă șirurile s1 și s2 sunt identice, o valoare negative dacă s1<s2 și o valoare pozitivă dacă s1>s2.
strchr(s1, ch);	Returnează un pointer la prima apariție a caracterului ch în șirul s1.
strstr(s1, s2);	Returnează un pointer la prima apariție a șirului s2 în șirul s1.

```
#include <iostream>
#include <string>
using namespace std;


int main()
{
    char greeting1[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };
    char greeting2[] = "Hello";
    cout << "Greeting message1: " << greeting1 << endl;
    cout << "Greeting message2: " << greeting2 << endl;


   cout << "\nLungimea sirului " << greeting1 << " este " << strlen(greeting1) <<
endl;


    char str[200];
    strcpy_s(str, 200, "Aceste ");
    strcat_s(str, 200, "siruri ");
    strcat_s(str, 200, "sunt ");
    strcat_s(str, 200, "concatenate.");
    cout << str << endl;


    cout << "\n Sirul " << greeting1 << " comparat cu " << greeting2 << " prin
strcmp returneaza " << strcmp(greeting1, greeting2) << endl;


    char str1[] = "Acesta este un sir de caractere de test";
    char * pch;
    cout << "\nSe cauta caracterul s in sirul \" " << str1 << " \" " << endl;
    pch = strchr(str1, 's');
    while (pch != NULL)
    {
```

```
        cout << "gasit la " << pch-str1+1<< endl;
        pch = strchr(pch + 1, 's');
    }

    pch = strstr(str1, "sir");
    strcpy_s(pch, 10, "fragment");
    cout << "\n" << str1 << endl;

    return 0;
}
```

- Clasa String din biblioteca <string>

Principalele metode implementate în cadrul acestei clase sunt:

- constructori: constructor fără parametri, constructor de copiere, constructori cu parametri (pe bază de subșiruri, pe bază de c-string-uri - secvențe de caractere terminate printr-un null, pe bază de buffer, constructor de umplere/ocupare, pe bază de gamă de valori, pe bază de listă de inițializare, constructor de transfer);

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s0("Sirul initial");

    // constructorii clasei String
    //constructor fara argumente
    string s1;

    //constructor de copiere
    string s2(s0);

    //constructori cu parametri
    // din s0 sunt preluate primele 3 caractere incepand cu pozitia 6
    string s3(s0, 6, 3);
    // din sirul "Secventa de caractere" sunt preluate primele 6
    string s4("Secventa de caractere", 6);
    // este preluat sirul dat ca parametru
    string s5("O alta secventa de caractere");
    // este multiplicat caracterul x de 10 ori
    string s6a(10, 'x');
    // este multiplicat caracterul * de 10 ori
    string s6b(10, 42);       // 42 = cod ASCII('*')
    // este preluat subsirul din s0 dintre pozitiile 0 si 5
    string s7(s0.begin(), s0.begin() + 5);
```

```
    cout << "s1: " << s1 << "\ns2: " << s2 << "\ns3: " << s3;
    cout << "\ns4: " << s4 << "\ns5: " << s5 << "\ns6a: " << s6a;
    cout << "\ns6b: " << s6b << "\ns7: " << s7 << '\n';
    return 0;
}
```

- destructor;
- operatorul de asignare.
- iteratori;

Metoda **begin()** returnează un iterator care pointează spre începutul șirului de caractere. Dacă șirul de caractere este definit de tip constant, atunci metoda returnează un **const_iterator**; altfel returnează un **iterator.** Ambele tipuri de iterator sunt de tip **random access iterator**.

Metoda **end()** returnează un iterator care pointează după terminatorul de șir și este recomandat ca acesta să nu fie dereferențiat. Metoda este folosită adeseori împreună cu begin() pentru a specifica un anumit interval în cadrul șirului de caractere. Dacă obiectul este un șir vid, atunci funcția returnează același lucru ca și begin().

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string str("String de test");
    for (string::iterator it = str.begin(); it != str.end(); ++it)
        cout << *it;
    cout << '\n';

    return 0;
}
```

- funcții pentru determinarea dimensiunii unui șir;

Funcțiile **size()** și **length()** sunt sinonime și returnează dimensiunea unui șir în octeți, corespunzând conținutului stringului, nefiind neapărat egală cu capacitatea sa maximă de stocare.

Funcția **resize()** este folosită pentru a redimensiona un string la lungimea de n caractere. Dacă valoarea n este mai mică decât dimensiunea actuală a șirului, aceasta va fi scurtată, șirul redimensionat va cuprinde doar primele n caractere, restul conținutului fiind eliminat. Dacă n este mai mare decât dimensiunea șirului curent, atunci pozițiile suplimentare din șirul redimensionat vor putea fi completate cu un caracter sau cu caracterul null, în funcție de lista de parametri.

Funcția **clear()** șterge conținutul unui string, acesta devenind vid (lungime=0).

Funcția **empty()** verifică dacă un string este vid(lungimea sa este 0). Returnează true dacă stringul este vid. Funcția nu alterează conținutul stringului.

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string str("String de test");
    cout << "Dimensiunea stringului str este de " << str.size() << " bytes.\n";
    cout << "Dimensiunea stringului str este de " << str.length() << " bytes.\n";

    string str1("Imi place sa scriu cod in C");
    cout << str1 << '\n';
    unsigned sz = str1.size();
    str1.resize(sz + 2, '+');
    cout << str1 << '\n';
    str1.resize(14);
    cout << str1 << '\n';

    char c;
    string str2;
    cout << "Introduceti textul pe linii. Introduceti caracterul (.) pentru a
finaliza:\n";
    do {
        c = cin.get();
        str2 += c;
        if (c == '\n')
        {
               cout << str2;
               str2.clear();
        }
    } while (c != '.');

    string content;
    string line;
    cout << "Introduceti textul. Introduceti o linie goala pentru a finaliza:\n";
    do {
        getline(cin, line);
        content += line + '\n';
    } while (!line.empty());
    cout << "Textul introdus este:\n" << content;

    return 0;
}
```

- funcții pentru accesarea elementelor unui șir;

Operatorul **[pos]** returnează referința la caracterul de pe poziția pos din cadrul stringului. Dacă pos este egală cu dimensiunea stringului și stringul este definit de tip constant, funcția returnează o referință la caracterul null '\0'. Stringul este indexat de la 0.

Metoda **at(pos)** returnează referința la caracterul de pe poziția pos din cadrul stringului. Funcția verifică automat dacă pos este o poziție validă (pos<dimensiunea stringului) și generează o excepție out of range în caz contrar.

Funcția **back()** este folosită pentru a accesa ultimul caracter dintr-un string. Funcția **front()** este folosită pentru a accesa primul caracter dintr-un string. Aceste funcții nu trebuie apelate pentru stringuri vide.

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string str("String de test");
    for (int i = 0; i<str.length(); ++i)
    {
        cout << str[i];
    }

    for (unsigned i = 0; i<str.length(); ++i)
    {
        cout << str.at(i);
    }

    str.back() = '!';
    cout << str << '\n';

    str.front() = 'T';
    cout << str << '\n';

    return 0;
}
```

- funcții supraîncărcate.

Operatorul +
Operatorii relaționali

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
       string str1 = "Hello";
       string str2 = "World";
       string str3;
       int  len;
```

```
        // copie str1 in str3
        str3 = str1;
        cout << "str3 : " << str3 << endl;

        // concateneaza str1 si str2 si depune rezultatul in str3
        str3 = str1 + str2;
        cout << "str1 + str2 : " << str3 << endl;

        // returneaza lungimea sirului concatenat
        len = str3.size();
        cout << "str3.size() :  " << len << endl;

        return 0;
}
```

3. Iteratorul

Iteratorul este un obiect care reținând adresa unui anumit element dintr-un set de elemente cum ar putea fi de exemplu un array are capacitatea de a parcurge elementele setului folosind un set de operatori cuprinzând minim un operator de incrementare ++ și un operator de dereferențiere *. Pointerul este și el o formă de iterator, dar nu toți iteratorii îndeplinesc aceleași funcționalități ca și pointerii.

```
#include <iostream>
#include <string>

using namespace std;

int main()
{

        cout << "Introduceti primul sir: ";
        string sir1;
        cin >> sir1;

        string::iterator si;
        for (si = sir1.begin(); si != sir1.end(); si++)
                cout << *si << endl;

        cout << "Introduceti al doilea sir: ";
        string sir2;
        cin >> sir2;

        sir1 += sir2;
        cout << sir1 << endl;
        string c("aaa");
        size_t found;

        found = sir1.find(c);
```

```
        if (found != string::npos)
                cout << "first " << c << " found at: " << int(found) << endl;
        else
                cout << "sirul nu a fost gasit" << endl;

        return 0;
}
```

5. Clasa Vector

Vectorii din biblioteca <**vector**> sunt structuri de date secvenţiale ce utilizează spaţii de memorie continue pentru elementele lor. Elementele unui vector pot fi accesate utilizând anumite deplasamente (vezi artimetica pointerilor). Vectorii îşi pot schimba în mod dinamic dimensiunea. Ocupă mai multă memorie comparativ cu tablourile uni-dimensionale alocate static pentru a putea administra într-un mod eficient stocarea elementelor.

```
#include <vector>
#include <iostream>
using namespace std;

int main()
{
        vector<int> myVector;

        myVector.push_back(5);
        myVector.push_back(6);
        myVector.push_back(7);

        for (int j = 0; j< myVector.size(); j++)
                cout << myVector[j] << " ";
        cout << endl;


        //afisarea unui vector folosind iteratori
        vector<int>::iterator i;
        for (i = myVector.begin(); i != myVector.end(); i++)
                cout << *i << " ";


        return 0;
}
```

4. Metoda sort din biblioteca <algorithm>

Headerul <**algorithm**> definește o colecție de funcții special create pentru a fi aplicate pe șiruri de elemente. Un șir este definit ca o secvență oarecare de obiecte ce pot fi accesate prin intermediul iteratorilor sau a pointerilor, cum sunt de exemplu array-urile și vectorii. Algoritmii

operează în mod direct prin intermediul iteratorilor asupra valorilor și nu afectează în nici un fel structura containerului (dimensiunea sau spațiul alocat acestuia).

```
template <class RandomAccessIterator>
void sort (RandomAccessIterator first, RandomAccessIterator last);
```

```
template <class RandomAccessIterator, class Compare>
void sort (RandomAccessIterator first, RandomAccessIterator last, Compare
comp);
```

Metodele **sort(first, last)** și **sort(first, last, comp)** sortează elementele din subșirul specificat prin doi iteratori (first, last) în ordine crescătoare. Elementele sunt comparate folosind operatorul < pentru primul exemplu și funcția comp pentru cel de-al doilea. Parametrii **first** și **last** sunt iteratori de acces aleator la pozițiile inițială și finală în secvența ce urmează să fie sortată. Subșirul pe care se va aplica sortarea va fi delimitat de cei doi iteratori. Iteratorii trebuie să pointeze către un tip de dată pentru care interschimbarea de elemente să fie definite. Parametrul **comp** este o funcție binară care primește ca argumente două elemente din intervalul specificat și returnează o valoare convertibilă la tipul bool. Valoarea returnată indică ordinea celor două argumente în șirul sortat.

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

bool functie(int i, int j)
{
	return (i<j);
}

struct clasa{
	bool operator() (int i, int j)
	{
		return (i<j);
	}
} obiect;

int main()
{
	int vi[] = { 32,71,12,45,26,80,53,33 };
	vector<int> v(vi, vi + 8);

	cout << "vectorul v initial:";
	for (vector<int>::iterator it = v.begin(); it != v.end(); ++it)
		cout << ' ' << *it;
	cout << '\n';

	sort(v.begin(), v.begin() + 4);
	sort(v.begin() + 4, v.end(), functie);
	sort(v.begin(), v.end(), obiect);

	cout << "vectorul v dupa sortare:";
	for (vector<int>::iterator it = v.begin(); it != v.end(); ++it)
```

```
                cout << ' ' << *it;
        cout << '\n';

        return 0;
}
```

5. Probleme propuse

6.1.

a) Scrieți o funcție care primește ca parametru un string și returnează numărul de litere mari (uppercase). Parcurgeti șirul de caractere utilizând indecși.

b) Scrieți o funcție care primește ca parametru un string și returnează numărul de cifre. Parcurgeti șirul de caractere utilizând iteratori.

6.2. Fie următoarea clasă:

```
class StudentAC
{
        string nume
        int nota;

public:
        StudentAC();
        StudentAC(string nume, int nota);
        void afisare();
        void modificareNota(int nouaNota);
};
```

a) Completați clasa cu definițiile metodelor deja declarate.
b) Scrieți și testați o funcție care primește ca parametru un vector<> de studenti și afișează informațiile fiecărui student din vector. Parcurgeți vectorul folosind iteratori.
c) Scrieți și testați o funcție care primește ca parametru un pointer la StudentAC care reprezintă un vector clasic, alocat dinamic. Funcția returnează un vector<> din std de studenți, cu același conținut ca și vectorul primit ca parametru.
d) Supraîncărcați operatorul de comparare din clasa StudentAC și creați un vector<> de studenți pe care ulterior să îl sortați în ordinea ascendentă a notelor, utilizând funcția sort din std.

7. Bibliografie

http://www.cplusplus.com/reference/cstring/

http://www.cplusplus.com/reference/string/string/
http://www.cplusplus.com/reference/vector/vector/
http://www.cplusplus.com/reference/iterator/iterator/
http://www.cplusplus.com/reference/algorithm/