Lab 7 – P2

Cuprins

1. Biblioteca standard C++

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

1. Biblioteca **funcțiilor standard** – conține funcții generice, de sine stătătoare, care nu pot fi încadrate în nicio clasă (este moștenită din C);
2. **Clasele orientate obiect** – o colecție de clase împreună cu funcțiile asociate acestora.

Biblioteca de funcții standard este impărțită 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.

Biblioteca de clase orientate obiect oferă suport pentru numeroase activități, precum intrare/ieșire, manipularea șirurilor de caractere, procesare numerică, etc. Aceasta include:

- Clase standard C++ de intrare/ieșire,
- Clasa string,
- Clase numerice,
- Algoritmi (generare, sortare, cautare, etc),
- Clase pentru containere (vectori, liste, stive, cozi, etc),
- Clase pentru lucrul cu iteratori,
- Clase pentru lucrul cu excepții,
- Clase pentru lucrul cu thread-uri, etc.

2. Funcţii standard de intrare/ieşire

Fişier header	Descriere
<iostream>	Acest header 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 header declară servicii utile pentru desfășurarea operațiilor de intrare / ieșire cu format prin intermediul manipulatorilor parametrizați cum ar fi **setw** și **setprecision**.
<fstream>	Acest header declară sevicii folosite pentru procesarea controlată a fișierelor.

```
#include <iostream>

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

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

	//exemplu de folosire cerr
	char stre1[] = "Mesaj eroare cerr";
	std::cerr << "Mesaj cerr : " << stre1 << std::endl;

	//exemplu de folosire clog
	char stre2[] = "Mesaj afisat cu clog";
	std::clog << "Mesaj clog : " << stre2 << std::endl;

	return 0;
}
```

Cele 3 stream-uri 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 stream-uri sunt direcţionate spre ieşirea standard (consola), dar ele pot fi redirecţionate 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 utilizînd funcția `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 stabilirea 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/.

3. Clasa Vector

Clasa Vector declarată în headerul `<vector>` este un template la clasă și reprezintă implementarea unor structuri de date secvențiale ce utilizează spaţii de memorie continue pentru elementele lor. La fel ca și în cazul array-urilor din C (un array este un tablou uni-dimensional alocat static (după exemplul celor folosite la programare în limbajul C) - de exemplu `int a[20]`), elementele unui vector pot fi accesate utilizând anumite deplasamente (vezi artimetica pointerilor). Vectorii utilizează spații de memorie alocate dinamic și își pot schimba în mod dinamic dimensiunea. Din această cauză, se pot întâlni cazuri când adăugarea unor elemente noi intru-un vector implică o realocare de memorie, urmată de o mutare a elementelor existente în zona nouă și eliberarea zonei de memorie veche. Vectorii ocupă mai multă memorie comparativ cu array-urile pentru a putea administra într-un mod eficient stocarea elementelor.

```
#include <vector>
#include <iostream>

int main()
{
	std::vector<int> myVector;

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

	std::cout << "Dimensiunea vectorului este " << myVector.size() << std::endl;

	std::cout << "Elementele vectorului sunt: ";
	for (unsigned int j = 0; j < myVector.size(); j++)
		std::cout << myVector[j] << " ";
	std::cout << std::endl;

	return 0;
}
```

În exemplul de mai sus am folosit două metode ale clasei Vector:

- `push_back()` – inserează un nou element la sfârșitul vectorului și actualizează dimensiunea acestuia;
- `size()` – returnează numărul de elemente ale vectorului;

Pentru o descriere completă a clasei vector și a metodelor sale, accesați linkul http://www.cplusplus.com/reference/vector/vector/

4. Iteratorul

Iteratorul este un obiect care, reținând adresa unui anumit element dintr-un container (de exemplu vector), are capacitatea de a parcurge elementele acestuia folosind un set de operatori, set ce conține măcar doi operatori și anume un operator de incrementare ++ și un operator de dereferențiere *.

Toate containerele din biblioteca standard implementează metodele `begin()` și `end()`, iar aceste metode returnează iteratori. Metoda `begin()` întoarce un iterator care corespunde începutului containerului (primul element poate fi accesat prin dereferențierea acestui iterator). Metoda `end()` returnează un interator care corespunde adresei de după ultimul element al containerului (se poate considera această adresă ca adresa de după container și **NU** reprezintă ultimul element din container și **NU** poate fi dereferențiat).

Pentru a declara un iterator specific unui container din biblioteca standard folosim următoarea sintaxă:

```
std::class_name<template_parameters>::iterator name;
```

De exemplu, pentru a declara un iterator pentru un vector de întregi folosim sintaxa:

```
std::vector<int>::iterator iIterator;
```

Putem folosi un iterator pentru parcurgerea elementelor unui vector, ca în exemplul următor:

```
#include <vector>
#include <iostream>

int main()
{
	int myInts[] = { 5, 6, 7 };
	std::vector<int> myVector(myInts, myInts + 3);

	std::cout << "Dimensiunea vectorului este " << myVector.size() << std::endl;

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

	std::cout << std::endl;

	return 0;
}
```

În exemplul de mai sus s-au folosit două metode ale clasei Vector:

- `vector (InputIterator first, InputIterator last, const allocator_type& alloc = allocator_type())` – constructor de inițializare (http://www.cplusplus.com/reference/vector/vector/vector/);
- `begin()` – returneaza un iterator la începutul vectorului;
- `end()` – returnează un iterator la o adresă de după sfârșitul vectorului.

5. Metoda `sort` din headerul `<algorithm>`

Headerul `<algorithm>` definește o colecție de funcții special concepute pentru a fi aplicate pe șiruri de elemente. Un șir de elemente este definit ca o secvență oarecare de obiecte ce pot fi accesate prin intermediul iteratorilor sau a pointerilor (de exemplu, un vector). Pentru lista întreagă a funcțiilor definite în headerul `<algorithm>` accesați http://www.cplusplus.com/reference/algorithm/

Algoritmii implementați cuprind o gamă variată de operații (căutare, sortare, numărare, etc.) și operează în mod direct prin intermediul iteratorilor asupra elementelor, fără să afecteaze în vreun fel structura containerului (dimensiunea sau spațiul alocat acestuia).

Metoda `sort(...)` poate fi folosită pentru sortarea unui șir de elemente și este definită în headerul `<algorithm>`, având prototipurile:

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

Parametrii funcției `sort` sunt:

`first, last`	Iteratori care indică începutul și sfârșitul secvenței de sortat. Șirul folosit este **`[first, last)`** și conține toate elementele între `first` și `last`, incluzând elementul reprezentat de `first` dar nu și elementul reprezentat de `last`.
`comp`	O funcție care primește două elemente de tipul elementelor din șirul de sortat ca argumente și returnează o valoare convertibilă la `bool`. Valoarea returnată indică dacă primul element trebuie să se regăsească înaintea celui de-al doilea în secvența sortată.

Exemplul următor ilustrează utilizarea metodei `sort(...)` pentru sortarea elementelor unui vector de întregi. În mod implicit, elementele vectorului vor fi sortate în ordine crescătoare.

```
#include <iostream>
#include <algorithm>
#include <vector>

int main() {
	int myInts[] = { 32, 71, 12, 45, 26, 80, 53, 33 };
	std::vector<int> myVector(myInts, myInts + 8);   // 32 71 12 45 26 80 53 33

	std::sort(myVector.begin(), myVector.end());     // 12 26 32 33 45 53 71 80

	std::cout << "Elementele vectorului sunt: ";
	std::vector<int>::iterator i;
	for (i = myVector.begin(); i != myVector.end(); i++)
	{
		std::cout << *i << " ";
	}
	std::cout << std::endl;

	return 0;
}
```

Dacă se dorește sortarea în ordine descrescătoare, sau în oricare altă ordine, trebuie folosit al doilea prototip al funcției `sort`, prototip ce necesită ca parametru un pointer la o funcție de comparare. Un exemplu de folosire a funcției `sort` ce utilizează acest al doilea prototip al funcției este:

```
#include <iostream>
#include <algorithm>
#include <vector>

bool myComp(int i, int j)
{
	return (i > j);
}

int main() {
	int myInts[] = { 32,71,12,45,26,80,53,33 };
	std::vector<int> myVector(myInts, myInts + 8);   // 32 71 12 45 26 80 53 33

	std::sort(myVector.begin(), myVector.end(),myComp);//80 71 53 45 33 32 26 12

	std::cout << "Elementele vectorului sunt: ";
	std::vector<int>::iterator i;
	for (i = myVector.begin(); i != myVector.end(); i++)
	{
		std::cout << *i << " ";
	}
	std::cout << std::endl;

	return 0;
}
```

6. Funcţii standard pentru manipularea şirurilor de caractere

C++ oferă două mecanisme pentru reprezentarea și manipularea șirurilor de caractere:

- Mecanismul moștenit din limbajul C;
- Mecanismul oferit de clasa string definită în header-ul `<string>`.

6.1. Reprezentarea şi manipularea şirurilor de caractere ca în limbajul C

La fel ca în limbajul C, în limbajul C++ șirurile de caractere sunt reprezentate sub forma unor tablouri unidimensionale de caractere reprezentate pe un singur octet și în care pe ultima poziție se află caracterul `'\0'`.

Index	0	1	2	3	4	5
Caracter	'H'	'e'	'l'	'l'	'o'	'\0'
Adresă (exemplu)	0x23452	0x23453	0x23454	0x23455	0x23456	0x23457

Există numeroase metode pentru inițializarea șirurilor de caractere:

```
char str1[] = "Hello";
char str2[6] = "Hello";
char str3[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
char str4[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };
char* str5 = "Hello";
```

La fel ca în cazul tablourilor unidimensionale alocate static (array-urile descrise anterior), nu este necesar să utilizăm tot spațiul alocat pentru memorarea șirului de caractere. De exemplu, putem aloca spațiu pentru 100 de caractere și să folosim doar o parte din acesta:

```
char str[100] = "Hello";
```

Funcțiile pentru manipularea șirurilor de caractere din biblioteca <string.h> din C au fost preluate și în limbajul C++, fiind definite în headerul <cstring> (http://www.cplusplus.com/reference/cstring/). În funcție de standardul C++ folosit, unele dintre funcțiile utilizate în C au fost declarate nesigure, iar folosirea lor va fi semnalată cu un mesaj de eroare. Aceste funcții au fost înlocuite cu altele considerate sigure și care au același nume ca funcțiile ce le înlocuiesc, dar cu sufixul "_s". Câteva dintre funcțiile cele mai utilizate sunt:

Funcția	Descriere
`strcpy(s1, s2);` `strcpy_s(s1, dim, s2);`	Copiază șirul s2 în s1.
`strcat(s1, s2);` `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 negativă dacă primul caracter care diferă are o valoare mai mică în s1 și o valoare pozitivă dacă primul caracter care diferă are o valoare mai mare în s1.
`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 <cstring>

int main()
{
	char greeting1[6] = { 'H', 'e', 'l', 'l', 'o', '\0' };
	char greeting2[] = "Hello";
	std::cout << "Lungimea sirului " << greeting1 << " este "
		<< strlen(greeting1) << std::endl;
	std::cout << std::endl << "Sirul " <<greeting1<<" comparat cu " << greeting2
	<<" utilizand strcmp returneaza "<< strcmp(greeting1, greeting2)<<std::endl;

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

	char str1[] = "Acesta este un sir de caractere de test";
	char* pch;
	std::cout << "\nSe cauta caracterul 's' in sirul \"" << str1 << "\" " <<
	     std::endl;
	pch = strchr(str1, 's');
	while (pch != nullptr)
	{
		std::cout << "gasit la " << pch - str1 << std::endl;
		pch = strchr(pch + 1, 's');
	}

	pch = strstr(str1, "sir");
	strcpy_s(pch, 10, "fragment");
	std::cout << std::endl << str1 << std::endl;
```

```
	return 0;
}
```

6.2. Headerul <string>

Clasa `string`, definită în headerul `<string>` oferă suport pentru reprezentarea și manipularea șirurilor de caractere. Astfel, instanțele acestei clase sunt obiecte ce reprezintă șiruri de caractere.

Constructorii clasei `string` sunt (C++11):

Constructor	Descriere
`string();`	Constructor implicit. Construiește un `string` gol (zero caractere).
`string(const string& str);`	Constructor de copiere.
`string(const string& str, size_t pos, size_t len = npos);`	Copiază subșirul din `str` care începe la poziția `pos` și are `len` caractere (sau subșirul care se termină la sfârșitul șirului `str`, dacă `str` este mai mic sau `len` este `string::npos`).
`string(const char* s);`	Copiază un șir de caractere reprezentat în varianta C.
`string(const char* s, size_t n);`	Copiază primele `n` caractere din șirul `s`.
`string(size_t n, char c);`	Crează un string din `n` copii ale caracterului `c`.
`template <class InputIterator> string(InputIterator first, InputIterator last);`	Copiază secvența de caractere din intervalul [first, last).
`string(initializer_list<char> il);`	Copiază caracterele din lista de inițializare.
`string(string&& str) noexcept;`	Constructor de mutare.

Un exemplu de folosire a acestor constructori este furnizat în exemplul următor:

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

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

	//constructorii clasei String:

	//constructor fara argumente
	std::string s1;
	std::cout << "s1: " << s1 << std::endl;

	//constructor de copiere
	std::string s2(s0);
	std::cout << "s2: " << s2 << std::endl;

	//constructori cu parametri
	//din s0 sunt copiate primele 3 caractere incepand cu pozitia 6
	std::string s3(s0, 6, 3);
	std::cout << "s3: " << s3 << std::endl;
```

```
	//din sirul "Secventa de caractere" sunt copiate primele 6
	std::string s4("Secventa de caractere", 6);
	std::cout << "s4: " << s4 << std::endl;

	//este copiat sirul dat ca parametru
	std::string s5("O alta secventa de caractere");
	std::cout << "s5: " << s5 << std::endl;

	//este multiplicat caracterul x de 10 ori
	std::string s6a(10, 'x');
	std::cout << "s6a: " << s6a << std::endl;

	// este multiplicat caracterul * de 10 ori
	std::string s6b(10, 42); // 42 = cod ASCII('*')
	std::cout << "s6b: " << s6b << std::endl;

	// este copiat subsirul din s0 dintre pozitiile 0 si 5
	std::string s7(s0.begin(), s0.begin() + 5);
	std::cout << "s7: " << s7 << std::endl;

	return 0;
}
```

Câteva dintre metodele și operatorii clasei `string`:

Metodă	Descriere
`operator=`	Operatorul de atribuire. Atribuie o nouă valoare string-ului, înlocuind conținutul actual.
`begin()`	Metodă ce returnează un iterator la primul caracter din string. Dacă șirul de caractere este definit de tip constant, atunci metoda returnează un const_iterator.
`end()`	Metodă ce returnează un iterator care indică adresa de memorie aflată după terminatorul de șir. Dacă obiectul este un șir vid, atunci funcția returnează același lucru ca și begin().
`size()` **`length()`**	Cele două metode sunt sinonime și returnează numărul de caractere al unui string.
`resize()`	Metodă folosită pentru a redimensiona un string prin trunchiere, dacă lungimea dorită este mai mică decât lungimea acestuia, sau completare cu caracterul null sau cu un caracter primit ca parametru, în caz contrar.
`clear()`	Metodă ce șterge conținutul unui string, acesta devenind vid.
`empty()`	Metodă ce verifică dacă șirul este vid (are lungimea 0).
`operator[size_t pos]`	Supraîncărcarea operatorului [] ce returnează referința la caracterul de pe poziția pos din șirul de caractere.
`at(size_t pos)`	Metodă ce returnează referința la caracterul de pe poziția pos din string. Funcția verifică automat dacă pos este o poziție validă (pos < dimensiunea stringului) și generează o excepție out of range în caz contrar.
`front();`	Metodă ce returnează referința la primul character din string.
`back();`	Metodă ce returnează referința la ultimul character din string.
`operator+`	Supraîncărcarea operatorului + ce concatenează două string-uri sau un string și un character.

Documentația acestor metode poate fi consultată accesând http://www.cplusplus.com/reference/string/string/.

Metodele și operatorii descriși anterior sunt folosiți în exemplul următor. Rulați programul și observați rezultatele obținute prin folosirea metodelor și a operatorilor menționați anterior:

```
#include <iostream>
#include <string>
int main()
{
	std::string str("String de test");

	// afisarea unui string
	std::cout << str << std::endl;

	// afisarea unui string folosind iteratoi
	for (std::string::iterator it = str.begin(); it != str.end(); ++it)
	{
		std::cout << *it;
	}
	std::cout << std::endl;

	// lungimea unui string
	std::cout << "Sirul \"" << str << "\" are lungimea " << str.size() << std::endl;

	// redimensionarea unui string
	str.resize(17, 'a');
	std::cout << "Sirul \"" << str << "\" are lungimea " << str.size() << std::endl;

	// verificarea daca un string este vid
	str.resize(0);
	if (str.empty())
	{
		std::cout << "Sirul \"" << str << "\" este vid. " << std::endl;
	}
	str = "String de test";

	// metoda at(pos)
	for (unsigned i = 0; i<str.length(); ++i)
	{
		std::cout << str.at(i);
	}

	// back()
	str.back() = '!';
	std::cout << std::endl << str << std::endl;

	// front()
	str.front() = 'T';
	std::cout << str << std::endl;

	std::string str1 = "Hello";
	std::string str2 = "World";
	std::string str3;

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

	return 0;
}
```

7. Masurarea duratei de execuţie folosind headerul <chrono>

```
#include <iostream>
#include <vector>
#include <chrono>
#include <cstdlib>
#include <algorithm>

using namespace std;

vector<int> generate_random_vector(int n)
{
	vector<int> vec;

	/* initialize random seed: */
	srand(time(NULL));

	while (n > 0)
	{
		n--;
		vec.push_back(rand());
	}

	return vec;
}

int main()
{
	int n = 100000;
	vector<int> vec = generate_random_vector(n);

	chrono::time_point<std::chrono::system_clock> start =
chrono::system_clock::now();

	sort(vec.begin(), vec.end());

	chrono::time_point<std::chrono::system_clock> end =
chrono::system_clock::now();

	std::chrono::duration<double> elapsed_seconds = end - start;
	std::cout << "Vectorul cu " << n << " elemente a fost sortat in " <<
elapsed_seconds.count() << " secunde." << endl;

	return 0;
}
```

8. Probleme propuse

1.
   - Scrieți o funcție care primește ca parametru un string și returnează numărul de litere mari (uppercase). Parcurgeti șirul de caractere utilizând indecși.
   - Scrieți o funcție care primește ca parametru un string și returnează numărul de cifre. Parcurgeti șirul de caractere utilizând iteratori.

2. Fie următoarea clasă:

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

- Completați clasa cu definițiile metodelor declarate.
- Scrieți și testați o funcție care primește ca parametru un `vector<>` de studenti și afișează informațiile fiecărui student din vector. Parcurgeți vectorul folosind iteratori.
- 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 biblioteca standard de studenți, cu același conținut ca și vectorul primit ca parametru.
- 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 biblioteca standard.

9. Bibliografie

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