Laborator 2 – Incompatibilităţi/Diferenţe între C şi C++
Completări C++

Supraîncărcarea (redefinirea) numelui de funcţii

În C nu este permisă existenţa a două funcţii care au acelaşi nume. În C++ acest lucru este posibil cu următoarea precizare: funcţiile nu pot să difere doar prin valoarea returnată, este obligatoriu ca lista de parametrii sa fie diferită (tipurile variabilelor din lista de parametri).

Exemplu: Să se scrie două funcţii una care adună două numere şi una care adună două şiruri de caractere

```
#include <iostream>
#include <string.h>
using namespace std;
int add(int a, int b){
        return a+b;
}
char * add(char *s1, char *s2){
        char *rez = new char[strlen(s1)+strlen(s2)+1];
        strcpy(rez, s1);
        strcat(rez, s2);
        return rez;
}
void main(){
        int a = 10, b =10;
        cout << "Adunare de numere " << add(a,b) << endl;
        char *s1="Primul ", *s2="laborator!";
        cout << "Adunare de caractere " << add(s1,s2) << endl;
}
```

Funcţii inline

Cuvântul cheie *inline* pus în faţa unei funcţii indică compilatorului să expandeze acea funcţie în momentul compilării, astfel încât codul obiect generat nu va conţine un apel de funcţie, ci va conţine codul obiect corespunzător acelei funcţii.

Obs:
1. inline este doar o "indicaţie", a cărei implementare diferă de la compilator la compilator.
2. Funcţile inline înlocuiesc cu succes macrourile construite cu #define şi evită neajunsurile acestor cauzate de expandarea codului sursa:
   - nu pot returna valori;
   - parametrii trebuie să fie precizaţi între paranteze, pentru o evaluarea conforma cu precedenţa operatorilor;
   - nu pot verifica tipul operanzilor.

Macrodefiniție	Funcție inline

#define Max(a,b) ((a)<(b)) ? (b) : (a)	inline int Max(int a, int b) {
	return (a<b) ? b : a ;
	}

```
#include <iostream>
#include <time.h>
```

```
using namespace std;
inline void swap_inline(int *a, int *b, int *c, int *d) {
    int temp; temp = *a; *a = *b; *b = temp; temp = *c; *c = *d; *d = temp;
}
void swap_call(int *a, int *b, int *c, int *d) {
    int temp; temp = *a; *a = *b; *b = temp; temp = *c; *c = *d; *d = temp;
}
int main() {
    clock_t start, stop;
    long int i;
    int a = 1, b = 2, c = 3, d = 4;
    start = clock();
    for(i = 0; i < 100L; i++)
        swap_inline(&a, &b, &c, &d);
    stop = clock();
    cout << "Timp apel functie inline: " << stop - start << endl;
    start = clock();
    for(i = 0; i < 900000L; i++)
        swap_call(&a, &b, &c, &d);
    stop = clock();
    cout << "\nDurata pentru functia apelata: " << stop - start << endl;
    return 0;
}
```

Alocarea dinamică a memoriei folosind operatorii new şi delete

Spre deosebire de limbajul C (în care managementul memoriei era implementat în biblioteci auxiliare şi era pus la dispoziţia programatorilor prin intermediul unor funcţii), în C++ au fost introduşi doi noi operatori pentru managementul memoriei:

- new – pentru alocarea memoriei cu sintaxa:
  tip _ptr =new tip
  tip _ptr =new tip(val_initializare)
  tip _ptr =new tip[n]

unde: *tip* – reprezintă tipul variabilei, care poate fi orice tip de date; *_ptr* – o variabilă de tip pointer; *val_initializare* – expresie cu a cărei valoare se iniţializează variabila dinamică. Rezultatul alocări poate fi un pointer spre tipul declarat sau valoarea NULL in caz de eşec.

- delete – pentru dealocarea memoriei cu sintaxa:
  delete _ptr

Aceşti operatori, pe lângă alocarea zonei de memoriei necesare, efectuează în plus operaţiile necesare iniţializării zonei respective, care sunt extrem de importante in cazul obiectelor.

Exemplu: Scrieţi un program care aloca memorie pană toata memoria este ocupată.

```
#include <iostream>
#include <new.h>
using namespace std;
void no_mem();
   int main(){
   int *ip;
   long dim;
   set_new_handler(&no_mem);
   cout << "Dimensiune bloc: "; cin >> dim;
   for(int i=1;;i++){
        ip = new int[dim];
        cout << "Alocare bloc i=" << i << endl;
   }
   return 0;
}
void no_mem(){
   cout << "Alocare imposibila \n";
```

```
    exit(1);
}
```

Operatorul rezoluţie (operator de acces)

In C++ este definit operatorul rezoluţie :: care permite accesul la un identificator cu domeniul fişier, dintr-un bloc în care acesta nu e vizibil din cauza unei operaţii.

Exemplu:

```
#include <iostream>
#include <string.h>
//using namespace std;
char str[20]="Sir global";
void fct(){
    char *str;
    str = "Sir local";
    std::cout << ::str << std::endl;
    std::cout << str << std::endl;
}
void main(){
    fct();
}
```

Tratarea excepţiilor

Tratarea excepţiilor permite tratarea problemelor care pot apărea într-un program într-un mod mai organizat. Avantajul tratări excepţiilor îl constituie o automatizare mai mare a codului.
Tratarea excepţiilor se realizează prin blocuri try…catch şi pot fi aruncate cu *throw*.

Sintaxă:

```
try{
….
}catch(tip1 arg){
…
}catch(tip2 arg){
}
.
.
.
catch(tipn arg){
…
}
throw exceptie;
```

Unde:
tip1,… ,tipn este un tip de date de bază sau o clasa exceptie este valoarea aruncată.
Blocul *try* va conţine secţiunea de cod unde se caută excepţiile, iar catch este folosit pentru a determina tipul excepţiei şi modul în care vor fi tratate excepţiile prinse.
Instrucţiunea *throw* se poate apela de oriunde din codul programului, într-un bloc *try…catch* sau funcţie. Dacă o eroare este aruncată şi nu este prinsă într-un bloc *try…catch* ea va determina o terminare anormală a programului prin apelul funcţiilor *abort*() sau *terminate*().

Exemplu: tratarea erorilor care pot apărea la împărţirea a două numere.

```
double impartire (double a, double b){
```

```
    if (b == 0){
        throw "Numitorul trebuie sa fie o valoare diferita de zero!";
    }
    return a/b;
}
int main()
{
    double a, b;
    cout << "a="; cin >> a;
    cout << "b="; cin >> b;
    cin.ignore();
    try{
        cout << "a/b = " << impartire(a, b);
    }catch(char * ex){
        cout << "Exceptia prinsa este: " << ex;
    }
    return 0;
}
```

Prinderea tuturor excepţiilor care apar într-un bloc se poate face printr-un bloc *catch*(…)

```
void fct(int i){
    try{
      if (i==0) throw 3;
      if (i==1) throw "O alta exceptie";
      if (i==2) throw 4.333;
    }catch(int a){
        cout << "Exceptie int! (" << a << ")\n";
    }catch (...){
        cout << "Restul de exceptii!\n";
    }
}
int main(){
    fct(0);
    fct(1);
    fct(2);
    return 0;
}
```

Cuvantul cheie auto

În versiunea C++11 s-a intrudus cuvântul cheie *auto* care specifica tipul unei variabile. Dacă o variabilă este declarată ca fiind auto tipul ei se stabileşte în funcţie de valoarea cu care este iniţializată. Dacă tipul de return al unei funcţii este *auto*, tipul valorii de return este determinat de tipul valorii returnate.

```
#include <iostream>
using namespace std ;
int main ( ) {
  auto a = 2 ; // declararea unei variabile de tip intreg
  auto b = 4 ; // declararea unei variabile de tip intreg
  auto proc ent = 3 . 6 ; // declarare a unei variabile de tip real
  double tablou [ ] = { 3.2 , 1.5 , 6.1 , 9 , 6.9 , 10 , 42};
  for ( auto i=a ; i<b ; i++) // declarare variabila de tip intreg
  cout << tablou [ i ] * procent << " " ;
  return 0 ;
}
```

Constanta nullptr

În versiunea C++11 s-a introdus cuvântul cheie nullptr, care indica spre un pointer NULL, nu constanta 0. Un exemplu de iniţializare a unui pointer double cu nullptr este următorul: double *d = nullptr;. Ambele variante de iniţializare cu valoare nula sunt acceptate, dar standadul de C++11 recomandă folosirea noii constante.

Probleme suplimentare

1. Să se scrie o funcţie care primeşte o dată calendaristică şi întoarce data calendaristica la care ne vom afla peste 5 zile (ex. 6.03.2012 → 11.03.2012). Folosind conceptul de supraîncărcarea a funcţiilor funcţia trebuie să fie capabilă să primească diferiţii parametri:
a. modificareData(„12/12/2012” ,…)
b. modificareData(„decembrie”, 12, 2012)
c. Propuneţi o altă formă de apel

2. Realizaţi un program care calculează distanţa dintre două puncte 2D, 3D. Pentru definirea unui punct 2D, respectiv 3D folosiţi structuri. Folosiţi supraîncărcarea funcţiilor pentru 3 funcţii care realizează acţiuni distincte (ex. funcţiile pentru calculul distanţei, afişare, etc).

3. Realizaţi un program care utilizează supraîncărcarea numelui de funcţii pentru a oferi funcţii care: adună un şir de numere întregi, adună un şir de numere reale, concatenează un şir de cuvinte. Pentru alocarea de spaţiu folosiţi două abordări (malloc() şi free(), new şi delete) .