Incompatibilități / Diferențe între limbajul C şi C++

1. Obiective

1. Operatorul de rezolutie
2. Operatorul new si delete
3. Functii inline
4. Functii care returneaza date de tip referinta
5. Apel prin referinta si apel prin valoare
6. Tipul de date boolean
7. Parametrii impliciti pentru functii
8. Tipul referinta
9. Supraincarcarea functiilor

2. Incompatibilități între C şi C++

2.1 Declararea şi definirea funcțiilor în C++

Varianta 1	Varianta 2

`#include "stdafx.h"`
`using namespace std;`
 	
`void dublare (int a)`
`{`
`cout<<a;`
`a=a*2;`
`cout<<" dublat este "<<a;`
`}`
 	
`void main()`
`{`
`//system("color fc");`
`dublare(10);`
`_getch();`
`}`
	`#include "stdafx.h"`
	`using namespace std;`
	
	`void dublare(int);`
	
	`void main()`
	`{`
	`dublare(10);`
	`_getch();`
	`}`
	
	`void dublare (int a)`
	`{`
	`cout<<a;`
	`a=a*2;`
	`cout<<" dublat este "<<a;`
	`}`

Ambele variante sunt corecte in C++.
Neapărat trebuie declarată funcția înainte de apelare.

2.2 Pointeri void

NU se face conversie implicita a tipului pointerilor "void*" in alte tipuri de pointeri.

Pentru tipul (void*) C++ admite conversia implicită în sensul

Tip pointer -> void*

Exemplu:

```
void * vp;
int *ip;
vp = ip; //corect in C si C++
ip=vp; //corect in C, incorect in C++
ip = (int*)vp; //corect in C si C++
```

3. Îmbunătățiri aduse de limbajul C++

Sugestii: 1) Sa se scrie numele de functii si de variabile cat mai explicit
Exemplu:

```
int countSpaces(char* str)
{
int counter;
// some computations…
return counter;
}
```

este mai clara si mai utilizata decat

```
int cnt(char* str)
{
int i;
return i;
}
```

2) Sa se adauge cat mai multe comentarii in scrierea codului
3) Nu puneti mai mult de 90 de caractere pe o linie!

3.1 Tipul boolean

Tipul boolean este un tip de baza in C++, C nu ofera acest tip.
Exemplu: program care returneaza true daca un element este gasit intr-un tablou si altfel returneaza false.

```
bool cauta()
{
bool gasit=false;
int i=0,n,elem, v[100];
cout<<"Introduceti lungimea vectorului "<<endl;
cin>>n;
while (i<n)
{
      cout<<"v["<<i<<"]= ";
      cin>>v[i];
      i++;
}
cout<<"Introduceti elementul pe care vreti
```

```
                    sa-l cautati in vector ";
cin>>elem;
for (i=0;i<n;i++)
{
if (elem==v[i])
 {
    gasit=true;
 }
}
return gasit;
}
```

3.2 I/O consola (cin, cout)

- folosim *cout* pentru printarea pe ecran

(cout – console output – stdout)

- folosim *cin* pentru retinerea unei intrari

(cin – console input – stdin)

Ambele sunt continute in biblioteca *iostream.* Pentru a le putea utiliza includem fisierul antet *iostream* si declaram folosirea spatiului de nume *using namespace std.*

3.3 Facilități la declarații

In C++ nu este impusa gruparea declaratiilor variabilelor locale la inceputul functiei, deci le putem declara oriunde in corpul functiei cu conditia sa le declaram inainte de a le utiliza!

Exemplu:

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

int suma (int a, int b)
{
return (a+b);
}

void main()
{
   int x,y,z;
   cout<<"Dati x si y "<<endl;
   cin>>x>>y;       //se citesc x si y
   cout<<"suma lui "<<x <<" cu "<<y<<" este "<<suma(x,y);
                    //se poate apela o functie direct in cout
     //asta doar in cazul in care functia nu e de tipul void
      z=23+suma(10,23);
      cout<<"z= "<<z;
   char nume[20];
      cout<<"Cum va numiti? "<<endl;
      cin>>nume;
      cout<<nume<<" este un nume foarte frumos!";

}
```

3.4 Variabile referință

In C++ putem declara identificatori ca referinte la obiecte de un anumit tip. Aceasta variabila referinta trebuie neaparat initializata la declarare cu adresa unei variabile pe care am definit-o deja.

```
void main()
{
int numar=10;
int &refint=numar; //referinta la int, trebuie neaparat initializat
cout<<"numarul este "<<numar<<" si refint = "<<refint<<endl;
refint=200; //automat numar=200;
cout<<"DUPA.. numarul este = "<<numar<<" iar refint = "<<refint<<endl;
}
```

Folosirea parametrilor formali referință în C++ permite transferul prin referință, deci permite renunțarea la folosirea pointerilor cand se transmit variabile.

Exemplu de interschimbare a două numere

Varianta C	Varianta C++

`void intersch(int *a, int *b)`
`{`
`int aux;`
`aux=*a;`
`*a=*b;`
`*b=aux;`
`}`
 	
`void main()`
`{`
`int a,b;`
`cout<<"Dati a si b "<<endl;`
`cin>>a>>b;`
`intersch(&a,&b);`
`cout<<"valorile interschimbate sunt "<<a <<" si "<<b;`
`}`
	`void intersch1(int &a, int&b);`
	
	`void main()`
	`{`
	`int a,b;`
	`cout<<"Dati a si b "<<endl;`
	`cin>>a>>b;`
	`intersch1(a,b);`
	`cout<<"valorile interschimbate sunt "<<a <<" si "<<b;`
	`}`
	
	`void intersch1(int &a, int&b){`
	`int auxiliar;`
	`auxiliar = a;`
	`a = b;`
	`b = auxiliar;`
	`}`

Dacă de exemplu avem

```
int variabila;
float &refvar=variabila;
```

tipurile referintei si a variabilei sunt diferite. De aceea compilatorul nu va crea o referinta la variabila de tipul int, ci va aloca memorie pentru o variabila in virgula mobilă, creand în memorie un obiect ascuns pentru referința dată.
ATENTIE!! Tipul variabilei cu tipul referintei la variabila respectivă trebuie sa fie acelaşi.

3 modalități de transmitere a parametrilor:

```
void functie_apel_prin_valoare(int x, int y, int z)
{
x=10;
y=20;
z=30;
}
void functie_apel_prin_pointer_referinta(int *x, int *y, int *z)
{
*x=10;
*y=20;
*z=30;
}
void functie_apel_prin_referinta(int &x, int &y, int &z)
{
x=10;
y=20;
z=30;
}
void main()
{
int x=10, y=20, z=30;
int &refx=x;
int &refy=y;
int &refz=z;
functie_apel_prin_valoare(x,y,z);
cout<<"Prin valoare "<<x<<" "<<y<<" "<<z<<endl;

functie_apel_prin_pointer_referinta(&x,&y,&z);
cout<<"Prin pointer "<<x<<" "<<y<<" "<<z<<endl;

functie_apel_prin_referinta(refx,refy,refz);
cout<<"Prin referinta "<<x<<" "<<y<<" "<<z<<endl;
}
```

3.5 Funcții care returnează referințe

```
struct carte
{
  char autor[64];
  char titlu[64];
  float pret;
};

carte biblioteca[3] = {
  {"Jamsa and Klander", "Totul despre C si C++", 49.95},
  {"Klander", "Hacker Proof", 54.95},
  {"Jamsa and Klander", "1001 Visual Basic Programmer's Tips", 54.95}};

carte& da_carte(int i)
 {
   if ((i >= 0) && (i < 3))
     return(biblioteca[i]);
   else
     return(biblioteca[0]);
```

```
 }

void main(void)
 {
   cout << "Pe punctul de a obtine cartea \n";
   carte& o_carte = da_carte(2);
   cout << o_carte.autor << ' ' << o_carte.titlu;
   cout << ' ' << o_carte.pret;
}
```

3.6 Parametri cu valori implicite

In C++ putem declara funcții cu valori implicite ale parametrilor. Acest lucru ne permite să apelăm funcția în mai multe moduri.

Argumentele cu valori implicite trebuie plasate la sfarşitul listei de argumente.

Exemplu:

```
int impartire(int a, int b=4)
{
int rezultat;
rezultat=a/b;
return rezultat;
}

void functie1(int, float=23.9, long=100);

int main()
{
      cout<<impartire(15)<<endl;
      cout<<impartire(20,10)<<endl;
      // cout<<impartire(); //incorect
      float f=4.5;
      functie1(11); //apel corect
      functie1(11,f);//apel corect
      functie1(11,2.6); //apel corect
      // functie(11,100); //apel incorect
return 0;
}

void functie1(int i, float f, long l)
{
cout<<"i= "<<i<<" f= "<<f<<" l= "<<l<<endl;
}
```

3.7 Supraîncărcarea numelui de funcții

In C++ este permisă existența a două sau mai multe funcții cu acelaşi nume cu precizarea ca parametri funcțiilor să fie diferiți şi valoarea returnată să fie diferită.
Exemplu:

```
#include "stdafx.h"
using namespace std;
```

```
   int operatie(int a, int b)
   {
   return (a*b);
   }

   float operatie (float a, float b)
   {
   return (a/b);
   }

   char * operatie(char *s1, char *s2)
   {
   char *rezultat = new char[strlen(s1)+strlen(s2)+1];
   strcpy(rezultat, s1);
   strcat(rezultat, s2);
   return rezultat;
   }

   int main ()
   {
   int x=5,y=2;
   float n=5.0,m=2.0;
   cout << operatie (x,y);
   cout << "\n";
   cout << operatie (n,m);
   cout << "\n";
   char *s1="Primul sir de caractere", *s2="Al doilea sir de
caractere";
cout<<"Adunarea a doua siruri de caractere "<<operatie(s1,s2)<<endl;
   return 0;
   }
```

3.8 Funcții inline

Sintaxa: inline tip nume (argumente ...)
{
instructiuni ...
}

Cuvantul cheie **inline** pus în fața unei funcții indică compilatorului să expandeze acea functie in momentul compilarii, astfel incit codul obiect genrat nu va contine un apel de functie, ci va contine codul obiect corespunzator acelei functii.
Obs: inline este doar o ‚indicatie', a carei implementare difera de la compilator la compilator (de ex: Borland C++ 3.1 va expanda functiile inline ce contin instructiuni repetitive (for, while), in timp ce VC++ 5.0 face acest lucru. De asemenea, exista cazuri in care compilatorul nu poate expanda functia inline. In aceste cazuri se va apela functia in mod normal. Avantajul functiilor inline este viteza sporita de executie. Se recomanda folosirea lor in cazul in care numarul de parametrii este redus, iar corpul functiei nu contine multe instructiuni.

Inlocuiesc cu succes macrourile construite cu #define si evita neajunsurile acestor cauzate de expandarea codului sursa:

- nu pot returna valori
- parametrii trebuie sa fie precizati intre paranteze, pentru o evaluarea conforma cu precedenta operatorilor.
Deasemenea permit o verificare de tip.

Ex:
#define Max(a,b) ((a)<(b)) ? (b) : (a)

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

Studiați următorul exemplu care afișează durata de execuție necesară pentru apelarea de 30000 de ori a fiecărei funcții

```
#include "stdafx.h"
#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;
 }


void main(void)
 {
   clock_t start, stop;
   long int i;
   int a = 1, b = 2, c = 3, d = 4;
```

```
  start = clock();
  for (i = 0; i < 300000L; i++)
    swap_inline(&a, &b, &c, &d);
  stop = clock();
  cout << "Time for inline: " << stop - start;

  start = clock();
  for (i = 0; i < 300000L; i++)
    swap_call(&a, &b, &c, &d);
  stop = clock();

  cout << "\nDurata pentru functia apelata: " << stop - start;
}
```

3.9 Operatori noi new şi delete

Spre deosebire de limbajul C (in care managementul memoriei era implementat in biblioteci auxiliare si era pus la dispozitia programatorilor prin intermediul unor functii), in C++ au fost introdusi doi noi operatori pentru managementul memoriei:
- new – pentru alocarea memoriei
- delete – pentru dealocarea memoriei

Acesti operatori, pe langa alocarea zonei de memoriei necesare, efectueaza in plus opeartiile necesare initializarii zonei respective, care sunt extrem de importante in cazul obiectelor.
Ex:

```
int main()
{
char * str = new char [100];
delete [] str;
}
```

Exemplu pentru alocarea memoriei pentru o matrice de 256 de octeți. Apoi programul completează matricea cu litera A şi ăi afişează conținutul:

```
void main(void)
 {
   char *array = new char[256];
   int i;

   for (i = 0; i < 256; i++)
     array[i] = 'A';

   for (i = 0; i < 256; i++)
     cout << array[i] << ' ';
 }
```

3.10 Operatorul de rezoluție

Exemplu:

```
#include "stdafx.h"
using namespace std;

int variabila_globala=300;

void main()
{
int variabila_globala=100;
cout<<"valoarea variabilei locale este "<<variabila_globala<<endl;
cout<<"valoarea variabilei globale este "<<::variabila_globala<<endl;
}
```

Cuvinte cheie in C++

asm	namespace
auto	new
bad_cast	operator
bad_typeid	private
bool	protected
break	public
case	register
catch	reinterpret_cast
char	return
class	short
const	signed
const_cast	sizeof
continue	static
default	static_cast
delete	struct
do	switch
double	template
dynamic_cast	this
else	throw
enum	true
except	try
explicit	type_info
extern	typedef
false	typeid
finally	typename
float	union
for	unsigned
friend	using
goto	virtual
if	void

inline int long mutable	volatile while xalloc

Obs: Cuvintele cheie nu pot fi folosite ca nume de identificatori (variabile, functii, clase etc.).

Operatori in C++

Operatorii limbajului C++ (de la cea mai inalta la cea mai scazuta prioritate) sunt:

::	Scope resolution	None
::	Global	None
[ ]	Array subscript	Left to right
( )	Function call	Left to right
( )	Conversion None	
.	Member selection (object)	Left to right
->	Member selection (pointer)	Left to right
++	Postfix increment	None
--	Postfix decrement	None
new	Allocate object	None
delete	Deallocate object	None
delete[ ]	Deallocate object	None
++	Prefix increment	None
--	Prefix decrement	None
*	Dereference	None
&	Address-of	None
+	Unary plus	None
-	Arithmetic negation (unary)	None
!	Logical NOT	None
~	Bitwise complement	None
sizeof	Size of object	None
sizeof ( )	Size of type	None
typeid( )	type name	None
(type)	Type cast (conversion)	Right to left
const_cast	Type cast (conversion)	None
dynamic_cast	Type cast (conversion)	None
reinterpret_cast	Type cast (conversion)	None
static_cast	Type cast (conversion)	None
.*	Apply pointer to class member (objects)	Left to right
->*	Dereference pointer to class member	Left to right
*	Multiplication	Left to right
/	Division	Left to right
%	Remainder (modulus)	Left to right
+	Addition	Left to right
-	Subtraction	Left to right
<<	Left shift	Left to right
>>	Right shift	Left to right
<	Less than	Left to right
>	Greater than	Left to right
<=	Less than or equal to	Left to right
>=	Greater than or equal to	Left to right

==	Equality	Left to right
!=	Inequality	Left to right
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
|	Bitwise OR	Left to right
&&	Logical AND	Left to right
||	Logical OR	Left to right
e1?e2:e3	Conditional	Right to left
=	Assignment	Right to left
*=	Multiplication assignment	Right to left
/=	Division assignment	Right to left
%=	Modulus assignment	Right to left
+=	Addition assignment	Right to left
-=	Subtraction assignment	Right to left
<<=	Left-shift assignment	Right to left
>>=	Right-shift assignment	Right to left
&=	Bitwise AND assignment	Right to left
|=	Bitwise inclusive OR assignment	Right to left
^=	Bitwise exclusive OR assignment	Right to left
,	Comma	Left to right

Obs:
a) Operatorii pot fi redefiniti la nivel global sau la nivel de clasa (overloading).
b) Urmatorii operatori NU pot fi redefiniti: . , .* , :: , ? : , # , ##.
c) Au fost introdusi operatori noi: new, delete, .*, ->,*.

4. Probleme

4.1 Afisarea in consola a unui text dintr-un fisier “test.txt”

```
int main()
{
//system("color fc");
char ch;
FILE* fp=fopen("test.txt", "r");
if(fp==NULL) {
cout<<"Eroare la deschiderea fisierului text" <<endl;
return 0;
}
while((ch=fgetc(fp))!=EOF)
cout<<ch;
fclose(fp);

_getch();
return 0;
}
```

4.2 Căutați de cate ori apare un cuvant într-un fişier.

4.3 Scrieți un program care sortează descrescător un sir de cuvinte. Cuvintele se vor citi de la tastatură. Pentru fiecare operație definiți câte o funcție.