Elemente noi, specifice limbajului C++

1. Operatorul de rezoluţie ::

- Operatorul de rezoluţie :: permite :
  - accesul la o dată globală redefinită local
  - accesul la o entitate aparţinând unui spaţiu de nume, calificând-o cu numele spaţiului urmat de operatorul de rezoluţie
  - accesul la o dată/funcţie membru ascunsă, calificând-o cu numele clasei urmat de operatorul de rezoluţie
- Operatorul :: are prioritate maximă

2. Spaţii de nume

- Un spaţiu de nume (namespace) permite crearea de noi domenii de vizibilitate (scope)
- Sintaxa declaraţie ***namespace*** este :

```
namespace nume    {
        listă_declaratii;
}
```

- Declaraţia ***namespace*** permite gruparea unor entităţi (clase, obiecte şi funcţii) sub un acelaşi nume. În acest fel, domeniul global poate fi divizat în "sub-domenii", fiecare cu numele său, evitându-se astfel posibile coliziuni.
- Semantica declaraţiilor într-un spaţiu de nume este similară declaraţiilor globale, exceptând faptul că domeniul numelor este restricţionat la spaţiul astfel construit.
- Cuvântul rezervat ***using*** se foloseşte pentru a importa un spaţiu de nume sau părţi ale acestuia în domeniul de vizibilitate curent.
- Toate fişierele din biblioteca standard C++ declară entităţile folosite în spaţiul de nume ***std*** . De aceea se utilizează directiva

  using namespace std;

  în toate programele ce folosesc entităţi definite în ***iostream***.

3. Sistemul I/O

- Sistemul de I/O din C asigură o interfaţă coerentă cu utilizatorul şi independentă faţă de echipamentul folosit.
- Forma abstractă de organizare se numeşte *stream*, iar instrumentul efectiv utilizat, fişier.
- Un stream este un concept abstract, o entitate logică, ce produce sau primeşte informaţii, reprezentând practic un termen general pentru orice flux de date
- În C++ suportul pentru sistemul de I/O este asigurat de o ierarhie de clase, ale căror declaraţii se găsesc in fişierul antet *iostream* (în variantele mai vechi, *iostream.h*)
- Sistemul de I/O din C++ operează atât cu date de tipuri predefinite, cât şi cu date de tipuri abstracte (obiecte)
- Atunci când începe execuţia unui program în C++, se deschid automat 4 streamuri incorporate

Stream	**Semnificaţie**	**Echipament implicit**
cin	intrare standard	tastatura
cout	ieşire standard	ecranul
cerr	ieşire pentru erori	ecranul
clog	versiune cu memorie tampon pentru erori	ecranul

- Operatorii >> şi << au un rol extins; pe lângă cel de deplasare pe biţi, ei sunt folosiţi ca operatori de I/O
  - >> operator de intrare (extracţie din stream)
  - << operator de ieşire (inserţie în stream)
- Operatorii de I/O se pot înlanţui şi pot fi supraîncărcaţi pentru tipurile de date abstracte

Observaţie : Anticipând, *endl* este un aşa numit manipulator ce se inserează ca o dată în *stream* şi semnifică trecerea la o nouă linie

Exemplul 1.

```
#include <iostream>

using namespace std;

int main(void)
{
    int a1,a2;
    double b,c;
    char d,e;
    char *s="Hello World!\n";
    cout<<s;
    cout<<"Tastati doi intregi ";
    cin>>a1>>a2;
    cout<<"Tastati doua nr. reale ";
    cin>>b>>c;
    cout<<"Tastati doua caractere  ";
    cin>>d>>e;
    cout<<"a1="<<a1<<", b="<<b<<", c="<<c<<", d="<<d<<", e="<<e<<endl;
    char sir[256];
    cout<<"Sir =";
    cin>>sir;
    cout<<"sir="<<sir<<endl;
    return 0;
}
```

4. Operatorul & şi tipul referinţă

- Construcţia &nume defineşte în C adresa de început a zonei de memorie alocată pentru nume
- În C++ operatorul & are şi rolul de a marca un nou tip derivat, **tipul referinţă**
- Tipul referinţă creează sinonime pentru nume deja existente (alias-uri)
- Spre deosebire de variabile şi pointeri, valoarea atribuită unei referinţe NU POATE FI SCHIMBATĂ; odată ce s-a atribuit o valoare unei referinţe aceasta rămâne asociată pentru toată durata execuţiei blocului de program
- O referinţă este în esenţă un pointer implicit
- Toate variabilele de tip referinţă trebuie iniţializate atunci când sunt create (cu excepţia cazurilor în care sunt membre ale unei clase, parametri de funcţii sau valori returnate de funcţii)
- Tipul referinţă poate funcţiona şi ca tip returnat de o funcţie, caz în care apelul de funcţie poate figura şi în stânga operatorului de atribuire
  Observaţie : în astfel de situaţii este important ca locaţia de memorie spre care se referă să nu fie eliberată la încheierea execuţiei funcţiei (de exemplu o referinţa către o variabilă locală nu se poate returna!)

5. Transferul prin referinţă al parametrilor

- Metoda implicită de transfer a parametrilor în C este cea prin valoare
- În C++ se implementează efectiv transferul prin referinţă al parametrilor, definind parametri formali de tip referinţă. Aceştia creează practic sinonime pentru parametrii efectivi.

6. Operatori de alocare şi dealocare dinamică a memoriei *new* şi *delete*

- În C se pot aloca zone de memorie în memoria heap folosind funcţii de bibliotecă (***malloc***, ***calloc***)
- C++ asigură un sistem propriu alternativ de alocare dinamică bazat de 2 operatori : ***new*** şi ***delete***
- În C++ alocarea dinamică de memorie se face prin intermediul operatorului unar ***new***
- Operatorul ***new*** returnează adresa de început a zonei de memorie alocată, sau pointerul NULL în cazul în care nu se poate face alocarea

  Observaţii : nu este necesară specificarea explicită a numărului de octeţi, ***new*** alocând memorie suficientă pentru a stoca obiectul de tipul specificat
  : ***new*** returnează automat un pointer spre tipul specificat, nu este necesară conversia explicită de tip

- Operandul operatorului ***new*** poate fi:
  - numele unui tip (predefinit sau definit de utilizator), de exemplu :
    new tip(expresie_de_iniţializare)
  - în cazul alocării de memorie pentru tablouri, numele tipului este urmat, între paranteze drepte de o expresie de tip întreg ce defineşte numărul de elemente ale tabloului, de exemplu :
    new tip[expresie].

  Prin aceasta, se rezervă o zonă de memorie de expresie****sizeof***(tip) octeţi.

  Observaţie : elementele unui tablou nu se pot iniţializa la alocare.
- O zonă de memorie alocată cu ***new*** se eliberează exclusiv prin intermediul operatorului ***delete***; în cazul eliberării memoriei alocate pentru tablouri sintaxa aferentă este ***delete*** []

  Observaţii : operatorul ***delete*** se poate folosi doar cu un pointer valid, alocat în prealabil prin utilizarea operatorului ***new***. Folosirea oricărui alt tip de pointer este o operaţie cu rezultat nedefinit.
- Un avantaj major al folosirii operatorilor ***new*** şi ***delete*** este posibilitatea supraîncărcării acestora.

7. Parametrii impliciţi ai funcţiilor

- Unii dintre parametrii formali ai unei funcţii pot avea valori iniţiale prestabilite
- Valorile implicite ale parametrilor se precizează doar în prototip
- Dacă numărul parametrilor de apel este mai mic decât cel din lista parametrilor formali, parametrii lipsă vor lua valorile implicite
- Începând cu primul parametru transmis implicit, toţi cei ce urmează vor fi introduşi ca având valori implicite

8. Supraîncărcarea funcţiilor (function overloading)

- Fiecare funcţie în C++ are un antet/prototip ce oferă următoarele informaţii
  1. numele funcţiei
  2. numărul argumentelor
  3. tipul argumentelor
  4. ordinea argumentelor
  5. tipul valorii returnate

- Fiecărei funcţii în C++ i se asociază o aşa-numită **semnătură**, care cuprinde doar **informaţiile 1-4**
- Supraîncărcarea (supradefinirea) funcţiilor este un concept de programare care permite programatorului să definească mai multe funcţii cu acelaşi nume în acelaşi domeniu de vizibilitate, cu condiţia ca acestea să aibă semnături diferite
- Supraîncărcarea funcţiilor reprezintă o formă rudimentară de polimorfism
- Compilatorul selectează versiunea corespunzătoare pe baza tipurilor argumentelor efective de apel
- Funcţiile supraîncărcate NU pot diferi numai prin tipul valorii returnate
- La apelul unei funcţii supraîncărcate se selectează versiunea corespunzătoare pe baza principiului adecvanţei maxime ("best-match"). Adecvat în acest context înseamnă că:
    - Există o potrivire exactă
    - Se poate face o conversie implicită, fără pierdere de informaţii
    - Se poate face o conversie, cu pierdere de informaţii
    - Există rutină de conversie definită de utilizator.
- În cazul în care nu se poate găsi o funcţie adecvată se generează un mesaj de eroare

9. Funcţii inline

- Funcţiile ***inline*** reprezintă o alternativă oferită de C++ macro-urilor din C
- Apelul unei funcţii ***inline*** se înlocuieşte la compilare prin corpul funcţiei (se expandează), eliminându-se astfel toate operaţiile specifice apelului şi respectiv revenirii din funcţiile obişnuite.
- Folosirea funcţiilor ***inline*** poate creşte viteza de execuţie a unui program cu preţul creşterii numărului de linii de program
- Compilatorul tratează directiva ***inline*** ca pe o indicaţie/sugestie, neexistând garanţia faptului că funcţia va fi efectiv ***inline***
- O funcţie ***inline*** nu poate fi recursivă şi nu poate fi referită printr-un pointer la funcţie (în astfel de situaţii indicaţia ***inline*** este ignorată de compilator)
- O funcţie declarată ***inline*** are implicit legătură internă
- Avantaje ale utilizării funcţiilor ***inline*** faţă de macro-uri rezultă din faptul că funcţiile inline sunt expandate de compilator, în timp ce macro-urile sunt expandate de preprocesor :
    - La apelul funcţiilor ***inline*** se face verificarea adecvanţei tipului parametrilor de apel
    - Expresiile folosite la apelul funcţiilor sunt evaluate o singură dată, în timp ce, în cazul macro-urilor, acestea pot fi evaluate de mai multe ori

10. Tipul *bool*

- Tipul de date logice ***bool*** este un tip predefinit în C++ şi se reprezintă pe 1 byte
- Datele de acest tip pot lua valorile ***true*** sau ***false***.
- Expresiile de condiţie au valori de tip *bool* în C++
- Între cele două valori există următoarele relaţii

!false == ***true*** şi respectiv ***!true*** == ***false***

11. Tipul structură

- Tipul structură este un tip definit de utilizator ce constă dintr-o mulţime finită componente, eventual de tipuri diferite
- Structurile se declară astfel :

struct nume {lista_elemente} lista_variabile;

unde nume este numele tipului structură introdus prin aceasta declaraţie, ce se poate folosi ca atare pe post de tip de date, lista_elemente cuprinde declaraţii ale componentelor tipului, separate prin punct şi virgulă, iar lista_variabile conţine variabilele de tipul structură introduse de declaraţie

- Accesul la componentele unei structuri se face folosind operatorul .(punct), atunci când se dispune de numele variabilei structură, sau cu operatorul ->, când se dispune de adresa variabilei structură

12. Tipul uniune

- Tipul uniune este un tip definit de utilizator ce constă dintr-o mulţime finită de componente, ce ocupă acelaşi spaţiu de memorie
- Uniunile se declară astfel :

union nume {lista_elemente} lista_variabile

unde nume este numele tipului uniune introdus prin aceasta declaraţie, ce se poate folosi ca atare pe post de tip de date, lista_elemente cuprinde declaraţii ale componentelor tipului, separate prin punct şi virgulă, iar lista_variabile conţine variabilele de tipul uniune introduse de declaraţie

- Accesul la componentele unei uniuni se face folosind operatorul .(punct), atunci când se dispune de numele variabilei uniune, sau cu operatorul ->, când se dispune de adresa variabilei uniune

13.Tipul enumerat

- Tipul enumerat este un tip definit de utilizator ce constă dintr-o mulţime finită de constante, numite enumeratori. O enumerare este deci formată dintr-un set de constante (cu reprezentare de tip întreg) care specifică astfel toate valorile posibile pe care le poate avea o variabilă de acel tip.
- Enumerările sunt declarate în mod asemănător structurilor :

    enum nume {lista_elemente} lista_variabile;

unde nume este numele tipului introdus prin aceasta enumerare, ce se poate folosi ca atare pe post de tip de date, lista_elemente cuprinde constantele tipului, separate prin virgule, iar lista_variabile conţine variabilele de tipul enumerat introduse de declaraţie

- Tipul enumerat este compatibil cu un tip întreg astfel ca fiecare constanta a tipului are asociată o valoare întreaga, primul enumerator fiind implicit asociat cu valoarea 0, dacă nu se specifică explicit altfel.
- Simbolurile folsite în tipurile enumerate NU pot fi introduse şi afişate direct.