Zombie Exterminator

C++ application

Cuprins…………………………………………………………………………………………1

Motivatie

Am ales sa fac acesta deoarece ……………….

# Limbajul C++

C++ este un limbaj de programare general, compilat. Este un limbaj mulţi-paradigmă, cu verificarea statică a tipului variabilelor ce suportă programare procedurală, abstractizare a datelor, programare orientată pe obiecte. În anii 1990, C++ a devenit unul din cele mai populare limbaje de programare comerciale, rămânând astfel până azi.

Bjarne Stroustrup de la Bell Labs a dezvoltat C++ (iniţial denumit C cu clase) în anii 1980, că o serie de îmbunătăţiri ale limbajului C. Acestea au început cu adăugarea noţiunii de clase, apoi de funcţii virtuale, suprascrierea operatorilor, moştenire multiplă, şabloane.

Limbajul C++ păstrează toate elementele limbajului C, beneficiind de eficientă şi flexibilitatea acestuia. Limbajul C++ este un superset al limbajului C. Incompatibilităţile sunt minore, de aceea, modulele C pot fi încorporate în proiecte C++ cu un efort minim.

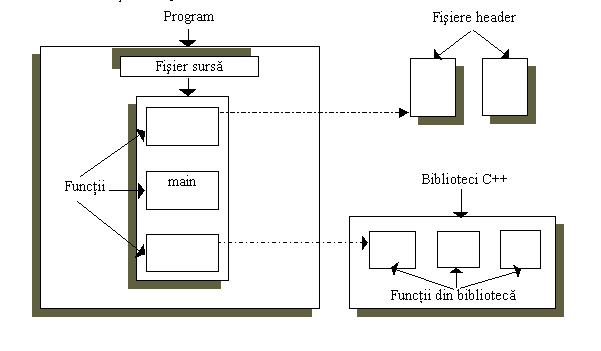
În 1985 a fost lansată prima ediţie a cărţii "The C++ Programming Language" (Limbajul de programare C++), oferind informaţii importante despre limbaj, care încă oficial. După ani de lucru, un comitet

ANSI-ISO a standardizat C++ în 1998 (ISO/IEC 14882:1998).

O dată cu evoluţia limbajului C++, a evoluat şi o biblioteca standard. Prima adăugire a fost biblioteca de intrări/ieşiri (I/O stream), care oferea facilităţi pentru a înlocui funcţiile tradiţionale C cum ar fi printf şi scanf. Mai târziu, printre cele mai semnificative adăugări la biblioteca standard a fost STL (Standard Template Library) (Biblioteca de formate standard).

Structura programelor C/ C++

Un program scris în limbajul C (sau C++) este compus din unul sau mai multe fişiere sursă. Un fişier sursă este un fişier text care conţine codul sursă (în limbajul C) al unui program. Fiecare fişier sursă conţine una sau mai multe funcţii şi eventual, referinţe către unul sau mai multe fişiere header.



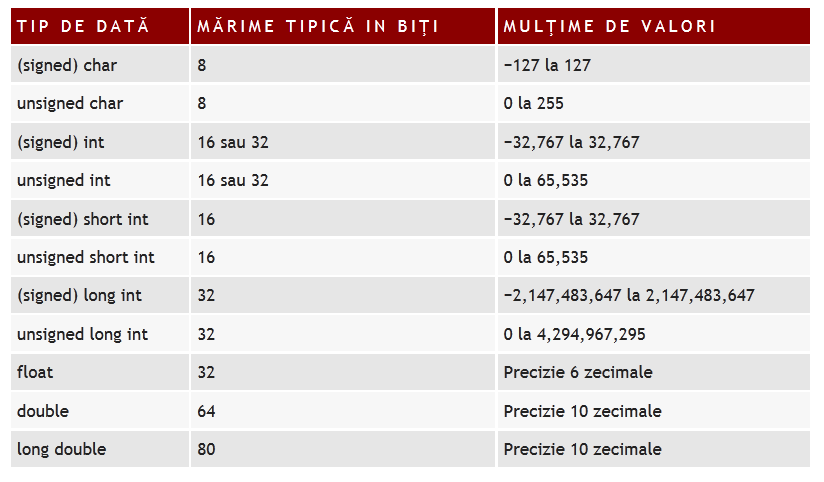
Funcţia principala a unui program este numită main. Execuţia programului începe cu execuţia acestei funcţii, care poate apela, la rândul ei, alte funcţii. Toate funcţiile folosite în program trebuie descrise în fişierele sursă (cele scrise de către programator), în fişiere header (funcţiile predefinite, existente în limbaj), sau în biblioteci de funcţii.

Un fişier header este un fişier aflat în sistem sau creat de către programator, care conţine declaraţii şi definiţii de funcţii şi variabile.

Cinci tipuri de date de baza în C În C există cinci tipuri de date de baza: caracter (char), întreg (int), virgulă mobilă (float), virgulă mobilă dublă precizie (double) şi fără nici o valoare (void). Toate celelalte tipuri de date din C se bazează pe cele cinci tipuri de baza.

Dimensiunea reprezentării şi domeniul valoric corespunzător pentru tipurile de date de baza pot să difere, în funcţie de tipul procesorului şi de modul de implementare a limbajului C. În toate cazurile, însă, un caracter se reprezintă pe un octet. Formatul exact al valorilor în virgulă mobilă depinde de modul lor de introducere. Valorile de tip char sunt, în general, utilizate pentru a memora valori definite de setul de caractere ASCII.

Mai jos este tabelul cu tipurile de dată fundamentale din C (definite de Standardul ANSI/ISO C):



Tipul void este folosit pentru a declara funcţii care nu returnează valori.

Modificarea tipurilor de bază

Exceptând tipul void, tipurile fundamentale pot fi precedate de diferiţi modificatori.

Modificatorul este folosit pentru a altera semnificaţia tipurilor de baza. Lista modificatorilor:

• signed

• unsigned

• long

• short

Puteţi aplică modificatorii signed, short, long şi unsigned tipului int. Puteţi aplică unsigned şi signed tipului char; long poate fi aplicat şi lui double.

Folosirea signed în declararea întregilor este redundanţa deoarece toate declarările int sunt considerate, implicit, signed. Acest modificator se foloseşte cu tipul char care este implicit unsigned.

Diferenţa dintre întregii signed şi unsigned constă în interpretarea diferită a bitului semnificativ (adică cel mai din stânga).

Pentru un întreg signed, compilatorul va consideră bitul semnificativ că fiind bit de semn. Dacă acest bit este 0 atunci numărul este pozitiv; dacă este 1 atunci este negativ.

Numerele negative sunt reprezentate în felul următor: se inversează toţi bîţîi (exceptând bitul de semn), se adaugă 1 (în binar) la număr, şi setează bitul de semn la 1. De exemplu, pentru numărul 32,767, reprezentarea binară este: 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1.

Dacă bitul de semn (aici este 0) este schimbat în 1, atunci numărul este interpretat că -1. Totuşi, dacă-l declaraţi că unsigned int, atunci numărul devine 65,535 când bitul de semn este setat la 1.

Cuvinte

Cuvintele cheie (keywords) sunt cuvinte care au un înţeles bine definit şi nu pot fi folosite în alt context.

Exemple:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| bool | Break | Case | Char | Const |
| Default | Double | Else | For | Float |
| If | Int | Long | Return | struct |
| Switch | Unsigned | While | Class | Namespace |

Variabile

O variabilă este un nume asociat cu o porţiune de memorie în care se memorează (stochează) o valoare, ce poate fi modificată ulterior (adică în timpul execuţiei) de către program. Toate variabilele trebuie declarate înainte de a fi folosite. Variabilele pot fi declarate în 3 locuri: în interiorul funcţiilor, în definiţia parametrilor formali, şi în afară tuturor funcţiilor.

Declararea variabilelor

Sintaxa:

tip\_dată nume;

unde:

•tip\_dată precizează tipul datei memorate în variabilă de memorie;

• nume este identificatorul variabilei de memorie.

Exemple:

|  |  |  |
| --- | --- | --- |
| Int a; | Char b; | Float e=1.234f; |
| Int x,z=1,y=2; | Double d=1.5484848484; | Char f=’a’; |

Comentarii

Pentru că un program să fie uşor de înţeles se folosesc comentariile.

Acestea sunt texte care vor fi ignorate de compilator, dar au rolul de a

explicită pentru programator anumite secvenţe de program.

// comentariu

sau

/\*comentariu

comentariu \*/

Formatarea

Marcajul unui bloc de instrucţiuni - se face prin acolade. Exemplu: { instrucţiuni; }

Separarea instrucţiunilor - că în marea majoritate a limbajelor de programare se face prin punct şi virgulă.

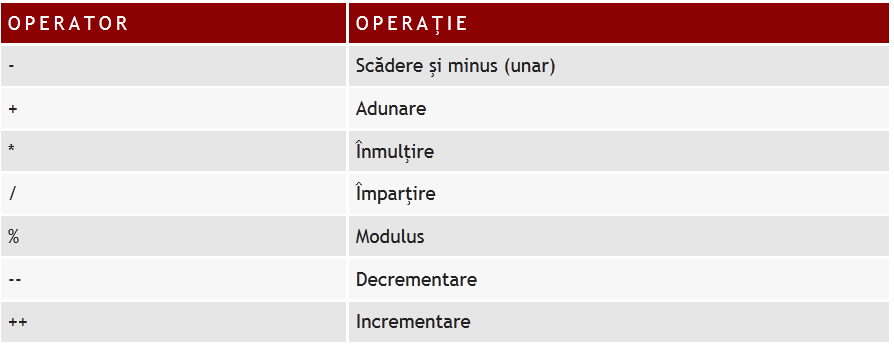
Operatori

Există 4 clase de operatori în C/C++: aritmetici,relaţionali, logici, sibitwise(pe biţi). Pe lângă aceştia mai sunt câţiva speciali, care îndeplinesc anumite sarcini.

Operatori Aritmetici

Operatorii de incrementare şi decrementare pot fi folosiţi atât că prefix cât şi sufix.

Când sunt folosiţi ca prefix, operaţia&nbsp;de incrementare sau decrementare se realizeazainainte de evaluarea expresiei, iar atunci când sunt folosiţi că sufix, operaţia de incrementare sau decrementare se realizează după evaluarea expresiei. Exemplu pentru ++, valabil şi pentru --.

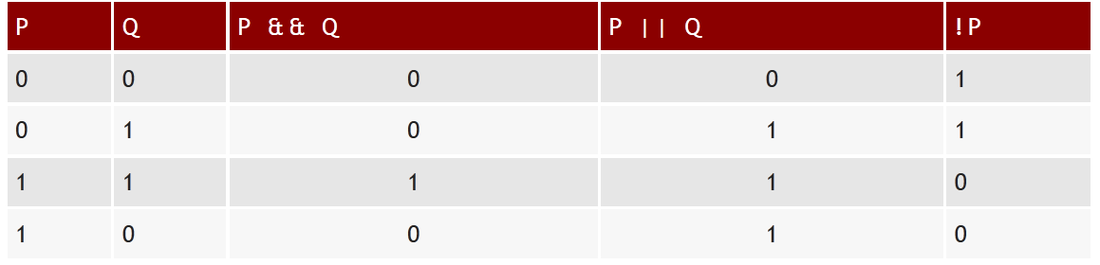


Operatori Logici şi Raţionali

În C, true este reprezentat de orice valoare diferită de zero, iar false este zero. În C++, se patreaza acest concept zero/non-zero, dar de asemenea este introdus un nou tip de dată, numit bool, cu valorile constante booleene: true şi false.

În C++ o valoare booleean 0 este automat convertită în false, iar o valoare non-zero este automat convertită în true.

Reciprocă este de asemenea valida.

Tabelul de adevăr pentru operatorii logici este prezentat mai jos, folosind 1 şi 0.

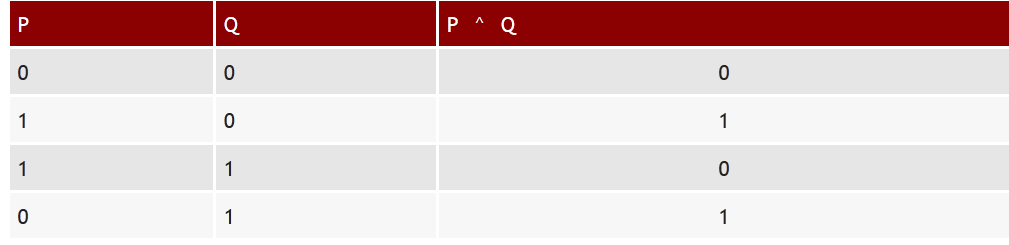
Tabelul cu operatorii relaţionali şi logici:



Întotdeauna puteţi folosi parantezele pentru a modifica ordinea efectuării operaţiilor într-o expresie.

Operatorii Bitwise

Realizează operaţii la nivelul biţilor. Pot fi aplicaţi numai tipurilor int şi char. Pe lângă operaţiile logice de mai sus, operaţiile cu biţi suportă şi operatorul Sau exclusiv (operaţia XOR).



Aşa cum se observă din tabel, rezultatul operaţiei XOR este true dacanumai unul din operanzi este true, altfel rezultatul este false.

Operatorii, constantele şi variabilele sunt constituentii expresiilor.

Scrierea (editarea) programului sursă.

Programele sursă sunt fişiere text care conţin instrucţiuni (cu sintactică şi semantică proprii limbajului utilizat). Programul (fişierul) sursă este creat cu ajutorul unui editor de texte şi va fi salvat pe disc (programele sursă C primesc, de obicei, extensia .c, iar cele C++, extensia .cpp).

Pentru a putea fi executat, programul sursă trebuie compilat şi linkeditat.

Compilarea

Procesul de compilare este realizat cu ajutorul compilatorului, care translatează codul sursă în cod obiect (cod maşină), pentru că programul să poată fi înţeles de calculator. În cazul limbajului C, în prima faza a compilării este invocat preprocesorul. Acesta recunoaşte şi analizează mai întâi o serie de instrucţiuni speciale, numite directive procesor. Verifică apoi codul sursă pentru a constată dacă acesta respectă sintaxa şi semantică limbajului. Dacă există erori, acestea sunt semnalate utilizatorului. Utilizatorul trebuie să corecteze erorile (modificând programul sursă). Abia apoi codul sursă este translatat în cod de asamblare, iar în final, în cod maşină, binar, propriu calculatorului. Acest cod binar este numit cod obiect şi de obicei este memorat într-un alt fişier, numit fişier obiect. Fişierul obiect va avea, de obicei, acelaşi nume cu fişierul sursă şi extensia .obj.

Linkeditarea

După ce programul sursă a fost translatat în program obiect, el este va fi supus operaţiei de linkeditare. Scopul fazei de linkeditare este acela de a obţine o formă finală a programului, în vederea execuţiei acestuia.Linkeditorul “leagă” modulele obiect, rezolva referinţele către funcţiile externe şi rutinele din biblioteci şi produce cod executabil, memorat într-un alt fişier, numit fişier executabil (acelaşi nume, extensia .exe)

Execuţia

Lansarea în execuţie constă în încărcarea programului executabil în memorie şi pornire execuţiei sale.

|  |
| --- |
|  |
|  | http://infoscience.3x.ro/c++/lectii/elemente%20de%20baza%20ale%20limbajului.files/image003.gif |

Structura generală a unui program C++

• un program C++ este constituit dintr-o succesiune de module, denumite funcţii

• una dintre aceste funcţii este funcţia principala, denumită main()

• main() este o funcţie specială, care trebuie să apară obligatoriu o singură dată în orice program C++

• execuţia oricărui program începe cu funcţia main()

• o funcţii este constituită din antet şi corp

• antetul funcţiei conţine numele funcţiei, tipul rezultatului pe care îl calculează funcţia şi o lista de parametri prin care funcţia comunica cu exteriorul ei, încadrată între paranteze rotunde

• corpul funcţiei conţine declaraţii şi instrucţiuni care specifică prelucrările realizate de funcţia respectivă

Instrucţiunile limbajului C++

Pentru a genera rezultatele dorite, un program trebuie să acţioneze asupra datelor într-un mod bine precizat. Descrierea acestor acţiuni se face cu ajutorul instrucţiunilor limbajului de programare.

Comenzile pe care programul le da calculatorului, atunci când programul este rulat se numesc instrucţiuni.

Instrucţiunile limbajului C++ sunt:

• instructiunea if;

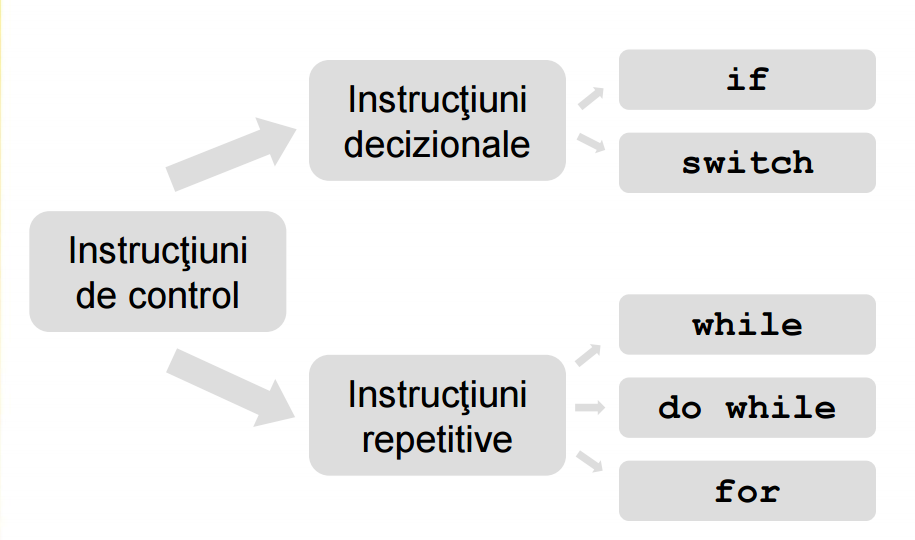
• instructiunea switch;

• instructiunea else;

• instructiunea while;

• instructiunea do while;

• instructiunea for.

Instrucţiunile se împart în două tipuri principale, decizionale şi repetitive că în graficul de mai jos:

Forma funcţiei main

.

Instructiunea return este utilizată pentru a încheia execuţia unei funcţii

şi a returna valoarea expresiei specificate în instructiunea return că valoare a funcţiei.

Structura generală a unui program C++

int main()

{

. . . .

return 0;

}

Un prim exemplu de program C++:

#include

void main( )

{

int a; // acesta este un comentariu de sfârşit de linie - am declarat o variabilă de tip întreg

cout << "Tastaţi un număr întreg ";

cin >> a;

cout << "Aţi tastat numărul " << a ;

}

Obs.: << este operatorul de inserţie

>> este operatorul de extracţie

cin este "console in" - pentru citire de date

Funcţii

O funcţie (subprogram) este un grup de instrucţiuni apelabil (invocabil) din alte părţi ale programului. De exemplu, main este o funcţie.

A apela o funcţie înseamnă a o execută.

Funcţiile ajută la modularizarea programului, la structurarea acestuia în unităţi logice.

O funcţie se declara în felul următor:

tip nume(lista\_parăm); // Prototipul funcţiei

Exemplu:

int suma(int a, int b);

lista\_parăm - lista de parametri, despărţiţi prin virgulă, este opţională.

Parantezele sunt obligatorii. Ele deosebesc funcţiile de alte entităţi C++ (de exemplu, variabile).

Tipul void reprezintă absenţa tipului. Cu alte cuvinte, funcţiile care nu returnează nicio valoare, au tipul void.

Funcţiile pot returna valori codului apelant (locul de unde a fost apelată funcţia) prin intermediul instructiuniireturn.

După declarare, o funcţie trebuie definită undeva în program.

#include

using namespace std;

int suma(int, int); // Declarare

int main()

{

int x, y;

cout << "Da-mi două numere întregi: ";

cin >> x >> y;

cout << "Suma lor este " << suma(x, y);

return 0;

}

int suma(int a, int b) // Definire

{

int rezultat = a + b;

// Puteam scrie direct

// return a + b;

return rezultat;

}

Când declaraţi o funcţie, puteţi omite numele parametrilor (aşa cum am făcut în exemplu), dar trebuie să precizaţi tipul lor.

Parametrii sunt variabile locale funcţiei (vizibile numai în blocul funcţiei).

Instructiunea return întrerupe execuţia funcţiei şi returnează valoarea expresiei, din dreapta, codului apelant.

Observaţi cum se apelează o funcţie: suma(x, y);. Dacă funcţia nu are parametri, se apelează doar cu nume + paranteze: funcFaraParam();.

Când execuţia unei funcţii se termină, controlul programului revine în punctul apelării şi programul îşi continuă execuţia normal.

Funcţiile void nu returnează valori, ele doar îndeplinesc o sarcina, deci nu pot avea instructiunea return.

Tablouri Unidimensionale

Tabloul este o structura de date internă formată dint-o mulţime ordonată de elemente, ordonarea făcându-se cu un ansamblu de indici. Tabloul de memorie se va identifica după un nume, iar fiecare element al sau, după numele tabloului şi numărul sau de ordine. Fiecare element al structurii se identifica după numele tabloului şi poziţia din tablou. De la început trebuie să se precizeze câte elemente are tabloul pentru că sistemul să-i aloce zona de memorie corespunzătoare. În timpul prelucrării tabloului nu i se pot adaugă mai multe elemente decât au fost alocate, pentru că se iese din spaţiul de memorie alocat. Tabloul de memorie este o structura de date statică.

Tabloul cu o singură dimensiune este numit vector. Declararea unui tablou de memorie de tip vector se face prin instructiunea:

Tip\_dată nume [ nr\_elemente ];

Aici, tip\_dată precizează tipul elementelor vectorului, nume este identificatorul vectorului, iar nr\_elemente este o constanţa întreagă care specifică numărul de elemente ale vectorului. De exemplu, prin:

int a[20];

se declara un vector cu 20 de elemente de tip întreg.

La declararea unui vector se pot atribui valori iniţiale elementelor sale astfel:

Tip\_dată nume[ nr\_elemente ] = { lista\_valori };

Exemplu:

int a[5]={10, 20, 2, 4, 9 };

În cazul declarării unui vector initializat se poate omite numărul elementelor sale, dacă se initializeaza toate elementele. Referirea la un element al vectorului se face prin construcţia:

nume[indice];

unde nume este numele tabloului, iar indice este numărul de ordine al elementului în vector. În C++ numerotarea indicilor se face pornind de la 0!

Pointeri

Pointerii reprezintă caracteristică cea mai puternică a limbajului de programare C++.

Pointerii sunt variabile care conţin adresa unei alte zone de memorie. Ei sunt utilizaţi pentru a date care sunt cunoscute prin adresa zonei de momorie unde sunt alocate.

Sintaxa utilizată pentru declararea lor este:

tip \*variabilă\_pointer;

Exemplu:

// declaraţie pointer

int \*p;

În lucrul cu pointeri se folosesc doi operatori unari:

• &: extragerea adresei unei variabile

• \*: referirea conţinutului zonei de memorie indicate de pointer (indirectare)

Exemplu:

// p ia adresa lui i p = &i;

p=&i;

// modificarea conţinutul zonei de memorie indicate de p

(\*p) = 6;

Referinţe

Referinţele, că şi pointerii, sunt variabile care conţin adresa unei zone de memorie. Semantic, ele reprezintă aliasuri ale unor variabile existente. Referinţele sunt legate de variabile la declaraţie şi nu pot fi modificate pentru a referi alte zone de memorie. Sintaxa folosită pentru declararea unei referinţe este:

Tip & referinţă = valoare;

Exemplu:

// declarative variabilă

int j=84;

// declaraţie referinţă

int& r = j;

Observaţie:

Obtinera adresei de memorie a unei variabile se poate face în următorul mod:

// declarare variablila

int x=42;

// obţinere adresa memorie

cout<<&x;

Clase

O clasa reprezintă o metodă logică de organizare a datelor şi funcţiilor unei aceleaşi familii. O clasa este declarată folosind cuvântul cheie class, care are o funcţionalitate similară cu cuvântului cheie struct din C. Diferenţa fundamentală este aceea că o clasa poate include între membrii săi şi funcţii, nu numai date că în structura struct din C. Formă generală a unei clase este:

class NumeClasa

{

...

declaraţii variabile membre

...

declaraţii funcţii membre

...

}

Specificatori de acces fac referinţă la permisiunea pe care membrii o dobândesc:

• private - membrii clasei sunt accesibili numai din cadrul altor membrii ai aceleiaşi clase sau

din clase prietene (friend)

• protected - membrii clasei sunt accesibili din cadrul membrilor aceloraşi clase sau clase

prietene (friend) şi, totodată, din membrii claselor derivate (derived)

• public - membrii sunt accesibili de oriunde clasa este vizibilă

Dacă se declara membrii unei clase înainte de a include o eticheta de permisiunea, aceştia vor fi

consideraţi (implicit) private.

Pentru a putea utiliza efectiv un tip de date (în cazul de faţă o clasa), trebuie să definim o variabilă de acel tip. Într-un mod similar declaraţiei

int i;

putem scrie:

NumeClasa obiect;

Accesarea membrilor unei clase se face prin specificatorul .

Exemplu:

// accesarea unei funcţii

obiect.FunctieMembra();

// accesarea unei variabile

obiect.variabilă = valoare;

Crearea şi distrugerea obiectelor

Clasa obiect;

În momentul definirii variabilei “obiect”, va fi alocat automat spaţiul de memorie necesar prin chemarea constructorului, acesta fiind eliberat la terminarea programului prin chemarea destructorului.

Constructori şi destructori

În general obiectele au nevoie de initializarea variabilelor lor sau de atribuirea dinamică a memoriei pe parcursul operaţiei de creare a acestora. Această pentru a deveni operaţionale şi a putea fi folosite fără surpriză de a întoarce rezultate neaşteptate (în general generatoare de probleme) pe parcursul execuţiei programului.

Dacă nu s-ar intializa s-ar obţine, probabil, un rezultat nedeterminat sau incorect.

Pentru a evita astfel de rezultate nedeterminate o clasa poate conţine o funcţie specială, denumită constructor. Această are acelaşi nume că şi clasa. Funcţia constructor este apelată automat când este creată o nouă instanţa (adică un nou obiect) al clasei respective - şi numai atunci.

Din punct de vedere cronologic, constructorul este apelat după alocarea memoriei necesare, deci în faza finală a creării obiectului, iar destructorul înaintea eliberării memoriei aferente, deci în faza iniţială a distrugerii sale.

Constructorii şi destructorii se declara şi se definesc similar cu celelalte funcţii membre, dar prezintă o serie de caracteristici specifice:

numele lor coincide cu numele clasei căreia îi aparţin; destructorii se disting de constructori prin faptul că numele lor este precedat de caracterul ~

• nu pot returna nici un rezultat

• nu se pot utiliza pointeri către constructori sau destructori

• constructorii pot avea parametri, destructorii însă nu. Un constructor fără parametri poartă denumirea de constructor implicit.

De remarcat este faptul că în cazul în care o clasa nu dispune de constructori sau destructori, compilatorul de C++ generează automat un constructor respectiv destructor implicit.

Exemplu:

// Constructorul unei clase Point

Point::Point()

{

// initializarea variabilelor la 0

x = 0;

y = 0;

}

// Destructorul unei clase Point

Point::~Point()

{

// dealocarea variabilelor şi a memoriei se face automat

}

# Blibliografie

* Doina Logofătu: *Algoritmi fundamentali in C++. Aplicații*, Ed. 1, Editura Polirom, Iași, 2007.
* http://informaticasalaoruandra.weebly.com/expresii.html
* Miloşescu M., Informatică. Manual pentru clasa a IX-a, Editura Didactică şi Pedagogică, Bucureşti, 2004
* Munteanu F., Programarea calculatoarelor. Manual pentru licee de informatică clasele X-XII, Editura Didactică şi Pedagogică, Bucureşti, 1994
* Popescu C., Culegere de probleme de informatică, Editura DonarisInfo, Sibiu, 2002
* Ministerul Educaţiei, Cercetării şi Tineretului, Centrul Naţional pentru Curriculum şi Evaluare în Învăţământul Preuniversitar, Proba scrisă la informatică. Examenul de bacalaureat – Variante (1-100) , Bucureşti2008
* http://www.cplusplus.com/
* http://ro.wikipedia.org/wiki/C\_(limbaj\_de\_programare)
* http://ro.wikipedia.org/wiki/C%2B%2B
* http://ro.wikipedia.org/wiki/Sintaxa\_limbajului\_C
* http://en.wikipedia.org/wiki/C%2B%2B\_Standard\_Library
* http://www.studytonight.com/cpp/introduction-to-cpp.php
* http://infoscience.3x.ro/c++.html
* http://info64.ro/