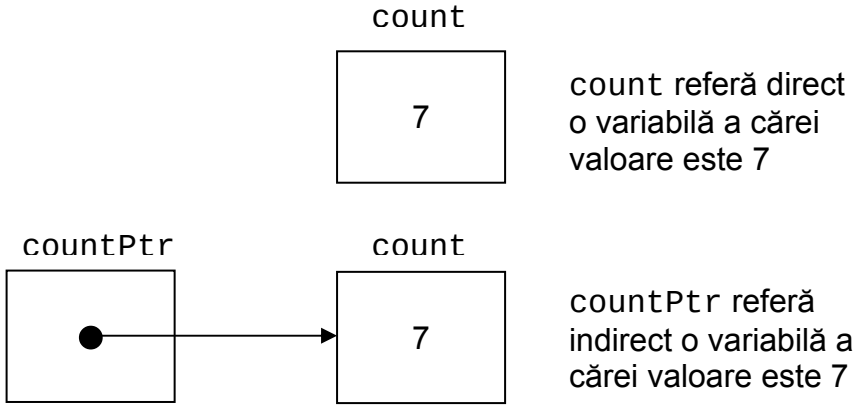
11. Pointeri

Pointerii reprezintă caracteristica cea mai puternică a limbajului de programare C++. În capitolele precedente am văzut cum se pot scrie funcții ale căror parametri sunt transmişi prin referință. Mecanismul transmiterii parametrilor prin intermediul pointerilor este o extensie a transmiterii prin referință. Am văzut, de asemenea, că tablourile sunt colecții de elemente de acelaşi tip care sunt stocate în locații succesive de memorie. Vom arăta în acest capitol că există o strânsă relație între pointeri şi tablouri şi vom studia modul în care se pot manipula tablourile prin intermediul pointerilor.

11.1 Variabilele de tip pointer

Variabilele de tip pointer stochează adrese de memorie. Pot, de exemplu, să păstreze adrese de memorie ale altor variabile care, la rândul lor, conțin alte valori. În acest sens, un nume de variabilă referă *direct* o valoare, iar un pointer referă *indirect* o valoare. Referirea unei valori printr-un pointer se numeşte *indirectare*.



Pointerii, ca orice altă variabilă, trebuie declarați înainte de a fi folosiți.

Exemplu

```
int *countPtr, count;
```

Prin aceste declarații, variabila `countPtr` este de tip `int*`, adică este pointer către o valoare întreagă. Variabila `count` este de tip întreg şi nu pointer la întreg. Fiecare variabilă declarată ca pointer este precedată de un asterisc `*`.

Exemplu

```
double *x, *y;
```

Atât `x` cât şi `y` sunt pointeri către valori de tip `double`. Aceste variabile pot păstra adrese de memorie ale unor valori de tip `double`. Pot fi declarați pointeri ca să pointeze către variabile de orice tip de dată.

Este indicat ca pointerii să fie inițializați fie odată cu declarația acestora, fie printr-o instrucțiune de asignare. Un pointer poate fi inițializat cu `0`, `NULL` sau cu o adresă de memorie. Un pointer cu valoarea `0` sau `NULL` nu pointează către nicio zonă de memorie. Constanta `NULL` este declarată în fişierul header `<iostream>` şi în alte câteva fişiere din biblioteca standard. Inițializarea prin valoarea `NULL` este echivalentă cu inițializarea prin valoarea `0`, dar în C++ se preferă cea de-a doua variantă. Întregul `0` este convertit automat către o adresă de tipul pointerului.

Valoarea `0` este singurul întreg care poate fi asignat direct unei variabile pointer fără o conversie prealabilă.

Pointerii constanți sunt cei al căror conținut nu se poate modifica. Un astfel de pointer trebuie inițializat în instrucțiunea de declarare.

Exemplu

```
int x;
const int *xPtr = &x;
```

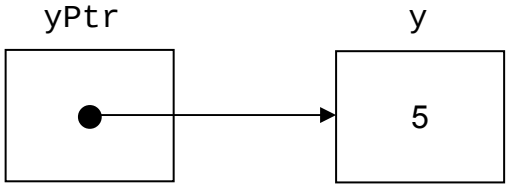
11.2 Operatori pentru pointeri

Operatorul adresă & este unar şi returnează adresa operandului său.

Exemplu

```
int y = 5;
int *yPtr;
yPtr = &y;
```

Prin ultima instrucțiune, adresa de memorie a variabilei `y` este încărcată în variabila pointer `yPtr`. În urma acestei asignări, vom spune că `yPtr` pointează către `y`.



Exemplu

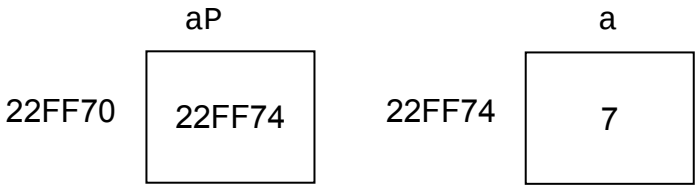
```
#include <iostream>
using std::cout;
using std::endl;

int main()
{
  int a;
  int *aP;
  a = 7;
  aP = &a;
  cout << "Adresa lui a este " << &a
       << "\nValoarea lui aP este " << aP;
  cout << "\n\nAdresa lui a este " << a
       << "\nValoarea lui *aP este " << *aP;
  cout << "\n\nOperatorii * si & sunt inversi unul altuia. "
       << "\n&*aP = " << &*aP
       << "\n*&aP = " << *&aP << endl;
  cout << "\n\nAdresa lui aP este " << &aP << endl;

  return 0;
}
```

Acest program afişează pe ecran următorul rezultat:

```
Adresa lui a este 0x22ff74
Valoarea lui aP este 0x22ff74
Adresa lui a este 7
Valoarea lui *aP este 7
```

```
Operatorii * si & sunt inversi unul altuia.
&*aP = 0x22ff74
*&aP = 0x22ff74
Adresa lui aP este 0x22ff70
```

În figura de mai jos reprezentăm variabila aP care presupunem că este localizată în memorie la adresa 22FF70 şi variabila a care se găseşte la adresa 22FF74.



Operandul operatorului adresă trebuie să fie un *lvalue* (*left value*), adică o entitate căruia îi poate fi asignată o valoare, de exemplu valoarea unei variabile. Operatorul adresă nu poate fi aplicat constantelor sau expresiilor al căror rezultat nu poate fi referit.

Operatorul * numit *operator de indirectare* sau de *dereferențiere* returnează un sinonim sau un alias pentru obiectul asupra pointează operandul său.

Instrucțiunea

```
cout << *aP << endl;
```

tipăreşte valoarea variabilei a care este 7, în acelaşi fel în care o face instrucțiunea

```
cout << a << endl;
```

Folosirea lui * în acest mod se numeşte *dereferențierea unui pointer*.

Un pointer dereferențiat poate fi folosit în partea stângă a unei instrucțiuni de asignare:

```
*aP = 5;
```

Prin această operație, valoarea 5 este asignată variabilei a.

- Un pointer dereferențiat poate fi folosit în diverse operații:

```
cin >> *aP;
```

Pointerul dereferențiat este un *lvalue*.

11.3 Transmiterea prin pointeri a parametrilor funcțiilor

În C++ se pot folosi pointerii şi operatorul de indirectare pentru a simula transmiterea parametrilor prin referință.

Exemplu

```
#include <iostream>
using std::cout;
using std::endl;

int cubPrinValoare(int);
void cubPrinReferinta(int*);

int main()
{
  int val = 5;
  cout << "Valoarea numarului este " << val;
  cubPrinValoare(val);
  cout << "\nValoarea dupa apelul cubPrinValoare(val) este "
```

```
          << val;

    val = 5;
    cout << "\n\nValoarea numarului este " << val;
    cubPrinReferinta(&val);
    cout << "\nValoarea dupa apelul cubPrinReferinta(&val) "
         << "este " << val << endl;

    return 0;
}

int cubPrinValoare(int n)
{
     return n * n * n;
}

void cubPrinReferinta(int *nPtr)
{
     *nPtr = *nPtr * *nPtr * *nPtr;
}
```

Acest program afişează:

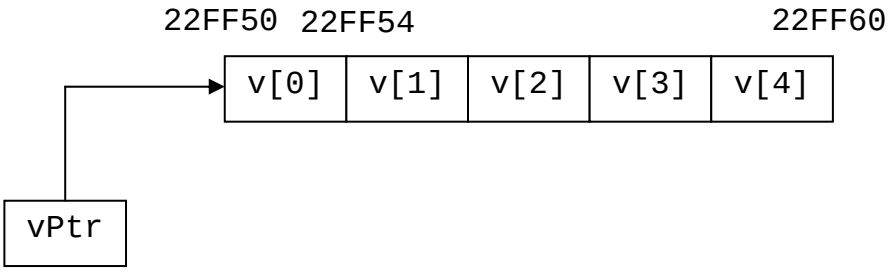
```
Valoarea numarului este 5
Valoarea dupa apelul cubPrinValoare(val) este 5

Valoarea numarului este 5
Valoarea dupa apelul cubPrinReferinta(&val) este 125
```

Mecanismul prin care valoarea parametrului actual `val` este modificată prin funcția `cubPrinReferinta(&val)` este similar celui prin care parametrul actual este modificat la transmiterea parametrilor prin referință. Se transmite adresa de memorie a variabilei care inițializează parametrul formal `nPtr`. Modificarea valorii la care pointează `nPtr` înseamnă modificarea valorii de la adresa `&val`.

11.4 Aritmetica pointerilor

Pointerii pot fi folosiți în expresii aritmetice, asignări şi comparații, însă nu toți operatorii pot avea pointeri ca operanzi. Asupra pointerilor pot fi realizate operații de incrementare (++), decrementare (`--`), adăugare a unui întreg (+ sau +=), scădere a unui întreg (`-` sau `-=`) şi scădere a unui pointer din alt pointer.

Să presupunem că declarăm tabloul `int v[5]` al cărui prim element este plasat de compilator la adresa de memorie `22FF50`. Pentru un calculator pe care întregii sunt reprezentați pe 4 bytes, cele cinci elemente ale tabloului sunt plasate la adresele de memorie din figura de mai jos.



Pointerul vPtr poate fi inițializat cu adresa primului element al taboului folosind una dintre instrucțiunile următoare:

```
int *vPtr = v;
vPtr = &v[0];
```

Adresa celui de-al doilea element al tabloului este &v[1].

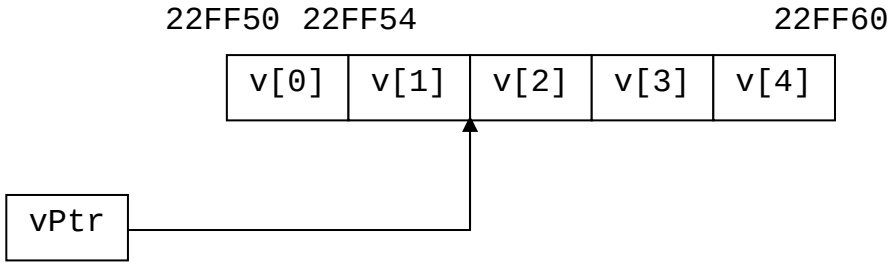
Spre deosebire de operațiile matematice în care adunarea 22FF50+2 are ca rezultat valoarea 22FF52, în aritmetica pointerilor adăugarea unui întreg la o adresă de memorie are ca rezultat o nouă adresă de memorie. Aceasta este egală cu adresa inițială la care se adaugă un număr de locații de memorie egal cu valoarea întregului înmulțită cu dimensiunea obiectului la care referă pointerul.

Exemplu

vPtr += 2;

are ca rezultat valoarea 22FF58, adică 22FF50 + 2 * 4 pentru că am presupus că întregii sunt stocați pe 4 bytes.

În urma acestei operații, vPtr va pointa către v[2].



Pentru un pointer către double, aceeaşi operație are ca rezultat valoarea 22FF60 pentru că la 22FF50 se adaugă 2 * 8 bytes.

Programul de mai jos prezintă operațiile care se pot efectua asupra pointerilor.

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

int main()
{
  int v[5];
  int *vPtr = &v[0];
  cout << "Adresa lui v[0] este " << &v[0] << endl;
  cout << "Adresa stocata in vPtr este " << vPtr << endl;

  vPtr += 2;
  cout << "\nAdresa stocata in vPtr dupa operatia vPtr += 2"
       << " este " << vPtr;

  vPtr -= 4;
  cout << "\nAdresa stocata in vPtr dupa operatia vPtr -= 4"
       << " este " << vPtr;

  vPtr++;
  cout << "\nAdresa stocata in vPtr dupa operatia vPtr++"
       << " este " << vPtr;

  ++vPtr;
  cout << "\nAdresa stocata in vPtr dupa operatia ++vPtr"
       << " este " << vPtr;
```

```
    vPtr--;
    cout << "\nAdresa stocata in vPtr dupa operatia vPtr--"
         << " este " << vPtr;

    --vPtr;
    cout << "\nAdresa stocata in vPtr dupa operatia –vPtr"
         << " este " << vPtr;

    int *v2Ptr = &v[4];
    cout << "\nRezultatul operatiei v2Ptr-vPtr este "
         << v2Ptr-vPtr << endl;

    return 0;
}
```

Acest program afişează următorul rezultat:

```
Adresa lui v[0] este 0x22ff50
Adresa stocata in vPtr este 0x22ff50

Adresa stocata in vPtr dupa operatia vPtr += 2 este 0x22ff58
Adresa stocata in vPtr dupa operatia vPtr -= 4 este 0x22ff48
Adresa stocata in vPtr dupa operatia vPtr++ este 0x22ff4c
Adresa stocata in vPtr dupa operatia ++vPtr este 0x22ff50
Adresa stocata in vPtr dupa operatia vPtr-- este 0x22ff4c
Adresa stocata in vPtr dupa operatia -vPtr este 0x22ff48
Rezultatul operatiei v2Ptr-vPtr este 6
```

Un pointer poate fi asignat altui pointer doar dacă cei dou au acelaşi tip. În caz contrar, trebuie aplicată o operație de conversie pentru ca pointerul din dreapta asignării să fie adus la tipul pointerului din stânga. Excepție de la această regulă în face pointerul `void*` care este un tip generic şi poate reprezenta orice tip de pointer fără a mai fi nevoie de cast. Pe de altă parte, pointerul `void*` nu poate fi dereferințiat pentru că numărul de bytes corespunzător lui nu poate fi determinat de compilator.

11.5 Pointeri şi tablouri

Tablourile şi pointerii sunt, în limbajul C++, în strânsă legătură. Un nume de tablou poate fi interpretat ca un pointer constant, iar pointerii pot fi indexați ca şi tablourile.

Pentru tabloul `v[5]` am declarat variabila pointer `vPtr` pe care am inițializat-o cu v, adresa primului element al tabloului. Elementul `v[3]` poate fi referit şi prin expresiile pointer

```
*(vPtr + 3)
*(v + 3)
```

Valoarea 3 din aceste expresii se numeşte *offset* la pointer, iar o astfel de expresie care accesează un element al unui tablou se numeşte *notație offset* sau *notație pointer*. Fără paranteze, expresia

```
*vPtr + 3
```

ar fi adunat valoarea 3 la expresia `*vPtr`, adică la `v[0]`.

Pentru pointeri se pot folosi indici la fel ca şi pentru tablouri.

Exemplu

```
cout << vPtr[1];
```

În schimb, numele unui tablou este un pointer constant şi nu poate fi folosit în operații care i-ar schimba conținutul.

Exemplu

```
v += 3;
```

este o operație invalidă pentru că ar modifica valoarea pointerului care reprezintă tabloul printr-o operație de aritmetică a pointerilor.

Tablouri de pointeri

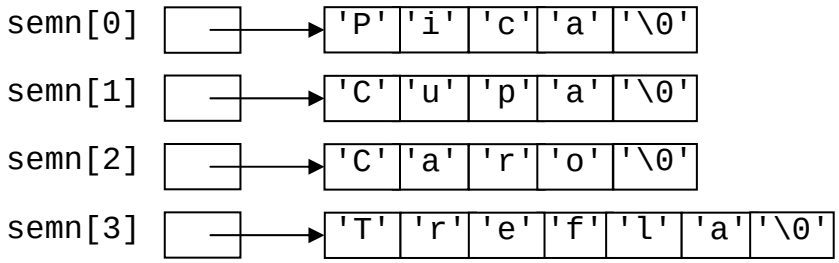
Tablourile pot conține pointeri. Se pot crea structuri de date care sunt formate din *string*-uri. Fiecare intrare într-un astfel de tablou este un string, dar în C++ un *string* este, de fapt, un pointer la primul său caracter. Astfel, fiecare intrare într-un astfel de tablou este un pointer către primul caracter al unui *string*.

Exemplu

```
const char *semn[4] = {"Pica", "Cupa", "Caro", "Trefla"};
```

Prin această instrucțiune, am declarat un tablou de patru pointeri la `char`.

Vom folosi tabloul `semn` pentru a reprezenta un pachet de cărți de joc.



Cele patru valori, `"Pica"`, `"Cupa"`, `"Caro"`, `"Trefla"`, sunt păstrate în memoria calculatorului ca şiruri de caractere terminate prin `NULL`. În tabloul `semn` sunt păstrate doar adresele de memorie ale primelor caractere din fiecare şir, iar şirurile au lungimi diferite. Dacă am fi optat pentru varianta unui tablou bidimensional în care fiecare rând reprezintă un tip şi fiecare coloană reprezintă o literă din fiecare tip, ar fi trebuit să fixăm numărul de coloane ale tabloului la dimensiunea maximă pe care o poate avea un şir. Pentru şiruri de dimensiuni mari, memoria neutilizată ar fi fost, astfel, destul de mare.

Vom ilustra folosirea tablourilor de *string*-uri prezentând un program care simulează amestecarea unui pachet de 52 de cărți de joc şi distribuirea acestora.

Folsim un tablou bidimensional cu 4 linii şi 13 coloane numit `pachet` pentru a reprezenta pachetul de 52 de cărți de joc.

		As	Doi	Trei	Patru	Cinci	Şase	Şapte	Opt	Nouă	Zece	Valet	Damă	Popă
		0	1	2	3	4	5	6	7	8	9	10	11	12
Pica	0													
Cupa	1													
Caro	2													
Trefla	3													

Linia 0 corespunde semnului "Pica", linia 1 corespunde semnului "Cupa", linia 2 corespunde semnului "Caro", iar linia 3 semnului "Trefla". Coloanele corespund valorilor înscrise pe fiecare carte. Coloanele de la 0 până la 9 sunt asociate cărților

de la as până la 10, coloana 10 este asociată valeților, coloana 11 damelor şi coloana 12 popilor. Numele semnelor vor fi păstrate în tabloul `semn`, iar valorile cărților vor fi stocate în tabloul `valoare`. Celula marcată, `pachet[1][4]`, reprezintă un cinci de cupă.

Acest pachet de cărți virtual poate fi amestecat astfel: Inițializăm tabloul pachet cu valori `0`. Alegem apoi la întâmplare o `linie` şi o `coloana`. Inserăm valoarea `1` în celula `pachet[linie][coloana]` pentru a arăta că aceasta este prima carte din pachetul amestecat. Continuăm acest proces inserând aleator valorile `2`, `3` ... `52` în tabloul `pachet` pentru a arăta care carte se va găsi pe poziția `2`, `3` ... `52`. Pentru că tabloul `pachet` se umple progresiv cu valori, este posibil ca în timpul derulării acestui algoritm o carte să fie selectată din nou. În acest caz, selecția este ignorată şi se alege un nou `linie` si o noua `coloana` până când este găsită o carte care nu a fost selectată.

Acest algoritm de amestecare a pachetului de cărți are dezavantajul că poate să se deruleze pentru o perioadă nedefinită de timp dacă este aleasă în mod repetat o carte care a fost deja selectată.

Pentru a împărți prima carte, căutăm în tabloul `pachet` celula `pachet[linie][coloana]` care conține valoarea `1`. Vom afişa, aşadar, numele cărții alese care va fi format din `semn[linie]` şi `valoare[coloana]`.

Algoritmul de implementare a soluției acestei probleme este următorul:

Inițializarea tabloului `semn`
Inițializarea tabloului `valoare`
Inițializarea tabloului `pachet`
Pentru fiecare dintre cele 52 de cărți
    Alege aleator o poziție din tabloul `pachet`

    Atăta timp cât poziția a fost deja aleasă
        Alege aleator o poziție din tabloul `pachet`

    Înregistrează numărul de ordine al cărții în poziția din tabloul `pachet`

Pentru fiecare dintre cele 52 de cărți
    Pentru fiecare poziție din tabloul `pachet`
        Dacă celula curentă din tabloul `pachet` *conține valoarea corectă*
            Tipăreşte numele cărții

Programul care implementează această problemă este următorul:

```
#include <iostream>
using std::cout;
using std::ios;
#include <iomanip>
using std::setw;

void Amesteca(int[][13]);
void Imparte(const int[][13], const char *[], const char *[]);

int main()
{
```

```
     const char *semn[4] = {"Pica", "Cupa", "Caro", "Trefla"};
     const char *valoare[13] =
          {"As", "Doi", "Trei", "Patru", "Cinci",
           "Sase", "Sapte", "Opt", "Noua", "Zece",
           "Valet", "Dama", "Popa"};
     int pachet[4][13] = {0};

     srand(time(0));

     Amesteca(pachet);
     Imparte(pachet, semn, valoare);

     return 0;
}

void Amesteca(int lPachet[][13])
{
     int linie, coloana;
     for(int carte=1; carte<=52; carte++)
     {
          do{
               linie = rand()%4;
               coloana = rand()%13;
          } while(lPachet[linie][coloana] != 0);

          lPachet[linie][coloana] = carte;
     }
}

void Imparte(const int lPachet[][13], const char *lSemn[],
             const char *lValoare[])
{
     for(int carte=1; carte<=52; carte++)
          for(int linie=0; linie<=3; linie++)
               for(int coloana=0; coloana<=12; coloana++)
                    if(lPachet[linie][coloana] == carte)
                         cout << setw(6) << lValoare[coloana]
                              << " de "
                              << setw(6) << lSemn[linie]
                              << (carte%2==0?'\n':'\t');
}
```

Programul afişează lista cărților după ce acestea au fost amestecate:

```
 Patru de   Pica        As de Trefla
  Noua de   Cupa     Valet de   Pica
   Doi de   Pica      Trei de   Pica
   Opt de   Cupa      Zece de   Caro
 Valet de   Caro     Cinci de   Caro
  Popa de   Pica      Trei de   Caro
  Popa de   Cupa       Opt de Trefla
    As de   Cupa      Sase de Trefla
  Popa de   Caro      Dama de   Cupa
```

```
   As de   Caro       Opt de   Pica
  Noua de Trefla     Valet de   Cupa
   Doi de Trefla      Noua de   Pica
 Patru de   Caro     Sapte de   Caro
  Trei de Trefla      Sase de   Caro
  Zece de   Pica        As de   Pica
 Valet de Trefla      Zece de   Cupa
  Dama de Trefla       Doi de   Caro
 Cinci de   Pica     Cinci de   Cupa
 Sapte de   Pica       Opt de   Caro
 Patru de Trefla     Sapte de   Cupa
 Cinci de Trefla      Sase de   Cupa
  Noua de   Caro     Patru de   Cupa
  Dama de   Caro      Sase de   Pica
   Doi de   Cupa      Trei de   Cupa
 Sapte de Trefla      Dama de   Pica
  Zece de Trefla      Popa de Trefla
```

Instrucțiunea

```
carte%2==0?'\n':'\t'
```

foloseşte operatorul ternar `?:` care are şablonul sintactic

condiție `?` *instrucțiune1* `:` *instrucțiune2*

Atunci când condiția este adevărată, se execută *instrucțiune1*, iar în caz contrar se execută *instrucțiune2*.