Tipul de date char

Tipurile de date pe care le-am prezentat în al doilea capitol au fost clasificate larg în: întregi și reale. Tipurile de date întregi prezentate au fost diferențiate doar prin numărul de octeți ocupați de variabilele: `short` și `unsigned short` – pe `2` octeți, `int` și `unsigned int` pe `4` octeți, `long long` și `unsigned long long` pe `8` octeți.

Există și tip de date întreg pe un octet și acesta este tipul `char`. Acum apare întrebarea de ce tipul `char` nu a fost prezentat deodată cu celelalte tipuri întregi. Motivul este că tipul de date `char` a fost gândit pentru ceva mai mult decât să stocheze valori întregi pe un octet. El permite în plus lucrul cu simboluri și mai departe stă la baza lucrului cu cuvinte.

În acest material ne ocupăm de regulile de care trebuie ținut cont pentru a înțelege cum manipulăm corect datele de tip `char`.

Fiind deci tip întreg pe un octet, avem tipurile:

`char`	Tip întreg cu semn deci valori cuprinse între -2^7 și 2^7-1, deci între -128 și 127 (inclusiv).
`unsigned char`	Tipul întreg fără semn deci valori cuprinse între 0 și 2^8-1, deci între 0 și 255 inclusiv.

Cele prezentate în continuare sunt valabile fie că discutăm despre tipul `char` fie că discutăm despre `unsigned char` (evident, cu excepția intervalului de valori pe care le poate lua o variabilă, cum e prezentat mai sus).

Iată câteva exemple:

Le considerăm în contextul declarațiilor:

```
char a, b;
int x;
```

`a = 10;`	Corect, atribuim o constantă întreagă unei variabile de tip întreg. Mai mult, valoarea constantei este în intervalul de valori posibile pentru variabilă.
`a = 10;` `b = a*2;`	Corect, de asemenea este vorba despre operații cu date întregi.
`x = a+b;`	Corect, de asemenea, se adună valorile din dreapta și rezultatul se copiază în variabila din stânga. Nu este nicio problemă din cauza faptului că este vorba de tipuri diferite (`int` respectiv `char`) deoarece știm că sunt permise astfel de situații.

Totuși, ce are totuși special tipul de date* `char`*?

Așa cum am spus mai sus, prin intermediul lui putem lucra în C/C++ cu simboluri și cuvinte. Prin convenție, fiecare simbol are asociat un cod numeric numit **cod ASCII** al său. Acest cod este valoarea numerică memorată când se stochează în RAM sau pe suport extern simbolul. Iată câteva coduri asociate simbolurilor:

`97` pentru `a`
`65` pentru `A`
`48` pentru `0`

32 pentru spațiu

De asemenea, celelalte litere ale alfabetului au coduri consecutive, adică b are 98, c are 99, B are 66, C are 67 etc. Observăm de pe acum și e bine să reținem că avem coduri mai mari pentru literele mici decât pentru literele mari.

Iată cele trei lucruri speciale pentru datele de tip char:

1. ***Constantele de tip char se pot scrie și prin punerea unui simbol între apostrofuri.*** Iată câteva exemple:

Considerăm declarațiile:

```
char a, b;
int x;
```

'a' 'C' '&' '2'	Constante de tip char corespunzătoare valorilor numerice: 97, 67, 38 și 50.
a = 'a';	Variabila întreagă a primește valoarea 97.
b = '2';	Variabila întreagă b primește valoarea 50.
x = 'a' + 2 * 'b';	Variabila x primește valoarea 97 + 2 * 98 adică 293.
a = 99;	Corect, variabila a primește valoarea codului simbolului c.
a = '2'*2;	Aici variabila a primește valoarea 100. Observăm diferența între 2 pus între apostrofuri, care este scris folosind sintaxa specifică la constantele de tip char - deci reprezintă valoarea 50 și 2 scris conform regulii obișnuite la constantele întregi.
'ab'	Eroare de sintaxă. Între apostrofuri se scrie un singur simbol.

În concluzie, scrierea unui simbol între apostrofuri reprezintă un alt mod de a specifica o valoare întreagă constantă (observăm că sunt permise toate operațiile cu numere întregi).

2. ***La afișarea cu* `cout` *a unei date de tip* `char`*, specificată explicit drept* `char`*, se afișează pe ecran simbolul ce are drept cod valoarea numerică memorată de data de tip* `char`*.***

Să explicăm mai exact la ce ne referim când spunem că data de afișat să fie indicată exact ca fiind de tip char: *variabilă de tip `char` dată ca parametru la afișare, constantă scrisă după regula simbolului între apostrofuri, expresie care are în față operatorul de conversie la `char` sau o funcție care returnează o dată de tip `char`.*

Trebuie să ținem cont de următorul lucru: dacă data de tip char apare într-o expresie alături de alte date și operatori, rezultatul se convertește automat la int chiar dacă rezultatul ar putea fi memorat pe un octet. Așa funcționează limbajul în cazul aplicării operatorilor, deci încearcă să facă toate calculele în tipul de date cel mai cuprinzător, iar în cazul în care acesta este int sau mai puțin de int, calculele se fac în int. În aceste cazuri rezultatul se consideră deci de tip int și se va tipări valoarea numerică în loc de simbol. Iată câteva exemple.

Considerăm declarațiile:

```
char a, b;
int x;
```

a = 'c'; cout<<a;	Se va tipări pe ecran c, fiind vorba de o varibilă de tip char dată ca parametru la cout.
a = 99; cout<<a;	Se va tipări, de asemenea valoarea c (exact ca la exemplul anterior). Nu are importanță modul în care a fost specificată constanta atribuită variabilei de tip char. În ambele cazuri ea stochează același luru iar lui cout îi este specificată o variabilă de tip char.

`cout<<'c';`	Se tipărește `c`. A fost pus parametru la cout o constantă specificată de tip `char`.
`cout<<99;`	Se tipărește `99`. Chiar dacă valoarea `99` se poate memora pe un octet ea nu este specificată lui `cout` efectiv după regula specială de scriere a constantelor de tip char, deci se va tipări valoarea numerică.
`cout<<(char)99;`	Se tipărește pe ecran `c`. Aici am specificat prin operatorul de conversie că valoarea finală a expresiei este de tip `char`.
`cout<<'a'+1;`	Se tipărește pe ecran `98`. Între constanta `char 'a'` și `1` se aplică operatorul `+` și am spus că în acest caz calculele se fac în `int`, așadar de tip `int` este și rezultatul final.
`cout<<'a'+'b';`	Este corect scris dar se afișează `195`. Este valabilă explicația de la exemplul anterior.
`cout<<(int)'0';`	Se afișează pe ecran `48`. Nu confundăm `0` scris între apostrofuri cu numărul `0`. Codul care se memorează pentru simbolul `0` este `48`. Noi înainte de afișare convertim valoarea la `int`.

`a = ' ';`
`x = a;`
`cout<<x;`
	Se afișează `32` (codul caracterului spațiu). Considerăm că între cele două apostrofuri de la prima atribuire am scris un spațiu.

3. ***La citirea cu `cin`, dacă parametru este o variabilă de tip `char`, ceea ce se introduce de la tastatură este interpretat ca simbol.***

Considerăm declarațiile:

```
char a, b;
```

`cin>>a;` `cout<<a;`	Introducem `d` apoi *enter*.	Se afișează pe ecran `d` întrucât în dreptul variabilei de tip char am introdus un simbol.
`cin>>a>>b;` `cout<<a<<" "<<b<<" "<<a+b;`	Introducem `aA` apoi *enter*.	Se memorează `97` (codul simbolului `a`) în variabila `a` și `65` în variabila `b`. Se tipărește `a A 162`
`cin>>a>>b;` `cout<<a<<" "<<b<<" "<<a+b;`	Introducem `1a` apoi *enter*.	Se memorează `49` (codul simbolului `1`) în variabila `a` și `97` în variabila `b`. Se tipărește `1 a 146`
`cin>>a;`	Introducem spațiu și apoi *enter*.	Nu se termină citirea. Este momentul să subiniem că întotdeauna citirea cu `cin` ignoră caracterele albe (spațiu, enter tab). Adică nu le consideră caractere utile pentru valorile variabilelor parametri. Așadar se asteaptă introducerea a cel puțin un caracter *nealb*.

Concluzia principală trebuie să fie că datele de tip `char` sunt stocate ca și numere, permit folosirea operatorilor aplicabili în general la datele întregi, dar la interacțiunea cu dispozitivele de intrare/ieșire, adică la citire/scriere, ele se interpretează ca simboluri. De asemenea, există varianta specială să indicăm constantele de tip `char` (simbol pus între apostrofuri).

Scrierea constantelor de tip `char` pentru catacterele speciale.

Pentru codurile de la `0` la `31` nu există specificate în standard simboluri asociate. Caracterele cu aceste coduri au în general alte specificații și nu tipărirea unui anume simbol pe ecran. De aceea, pentru ele nu putem folosi scrierea specifică la constante pentru că pur și simplu nu avem ce pune între apostrofuri ca simbol. Există totuși o convenție care ne permite să ne referim și la ele ca și constante de tip `char`: *secvențele escape*.

O secvență escape se specifică prin mai multe simboluri între apostrofuri, primul simbol fiind `\` (*backslash*). Chiar dacă se scriu mai multe simboluri nu trebuie să ne gândim că ar fi vorba despre mai multe caractere, este vorba despre unul singur dar indicat într-un anume fel.

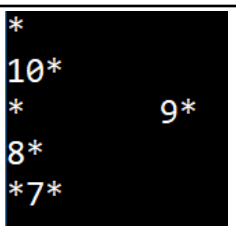
Iată secvențele escape pentru câteva caractere speciale:

`'\n'`	Codul ASCII `10`, caracterul *enter*. Afișatea acestui caracter are ca efect mutarea cursorului de pe ecran pe rândul următor. Noi de fapt am mai folosit acest carcter ca parte a mesajelor. Efectul de mutare a cursorului la începutul rândului următor apare și dacă îl tipărim ca pe un caracter izolat și dacă îl scriem în componența unui șir de caractere (adică între ghilimele).
`'\t'`	Codul ASCII `9`, caracterul *tab*. Tipărirea lui ca și simbol are ca efect apariția pe ecran a unu "spațiu mai lung".
`'\b'`	Codul ASCII `8`, caracterul *backspace*. Tipărirea lui are ca efect mutarea cursorului o poziție în stânga, adică sub ultimul simbol tipărit pe ecran pe linia curentă. Nu îl șterge de pe ecran dar el va dispărea, fiind suprascris, la următoarea afișare.
`'\a'`	Codul ASCII `9`, caracterul *alarm*. Tipărirea lui are ca efect un sunet care se aude din speaker sau boxe.

Dintre aceste caractere, *enter* este des folosit, la celelalte le-am prezentat efectul poate mai mult ca amuzament.

Dacă rulăm următorul cod, rezultatele se vor vedea în captura de consolă afișată în dreapta.

```
cout<<"*"<<'\n'<<(int)'\n'<<"*\n";
cout<<"*"<<'\t'<<(int)'\t'<<"*\n";
cout<<"*"<<'\b'<<(int)'\b'<<"*\n";
cout<<"*"<<'\a'<<(int)'\a'<<"*\n";
```



Pe fiecare linie de cod noi am scris tipărirea unui caracter ca "simbol" urmat de codul ASCII al său, cele două încadrate de doua caractere *stea*.

Să explicăm ce este în dreapta. Caraterul *enter* mută cursorul pe linia următoare, deci pe prima linie rămâne doar o steluță, iar pe a doua, de la început este codul lui *enter* (`10`) și steluța de final a primei linii de cod.

Noi am tipărit final de linie la fiecare instrucțiune deci pentru următorul simbol se trece pe rând nou. Acolo observăm un spațiu mare între prima steluță și `9`, adică s-a afișat "simbolul" *tab* urmat de codul său.

Pe linia următoare a ecranului sunt cele produse de a treia linie de cod. Steluța de început s-a afișat, dar tipărirea "simbolului" *backspace* a dus cursorul sub ea. Apoi, la afișarea codului lui *backspace* (`8`) acesta a suprascris-o. Pe ultima linie apare doar `7` între steluțe deoarece efectul de tipărire a simbolului este de fapt sunetul produs.

Alte trei secvențe escape particulare se pot folosi pentru caracterele *apostrof*, *ghilimele* și *backspace*. Ele se scriu precedate de *backslash*.

`'\''`	Caracterul apostrof.
`'\"'`	Caracterul ghilimele.
`'\\'`	Caraterul backslash.

Cum ne referim ca și caractere la celelalte ce au coduri între `0` și `31`? Noi aici am prezentat doar `4` dintre caracterele neafișabile. Răspunsul vine din forma generală de scriere a secvențelor escape: între apostrofuri, cu backspace în față, putem scrie un cod în baza de numerație `8`. Adică scrierea în baza `8` a codului ASCII. În acest mod putem indica toate caracterele, inclusiv pe cele cu simbol asociat, ca secvențe escape.

Exemple:

`cout<<'\141';`	Transformarea valorii `141` din baza `8` în baza `10` are ca rezultat `97`. Așadar se va tipări pe ecran simbolul `a`, fiind vorba despre indicarea efectiv a unei constante de tip `char`.
`cout<<'*'<<'\12'<<'*';`	Se tipăresc două steluțe, însă fiecare pe câte un rând. Este ca și cum în loc de `'\12'` am fi scris `'\n'` deoarece `12` este scrierea în baza `8` a lui `10` (codul ASCII al lui *enter*).
`cout<<'\n' + 'a' + '\11';`	Evident că nu sunt decât rațiuni didactice să scriem așa. Acum ne ajută doar la o mai bună fixare a cunoștințelor. Este vorba despre constante de tip `char`, indicate în diverse moduri, cu operatori aritmetici. Deci calculele se fac în `int`, adunându-se numerele ce reprezintă codurile lor ASCII. Rezultatul afișat este `116` (`10 + 97 + 9`).

Exerciții și probleme rezolvate

1. Având memorat în variabila `x` de tip `char` codul unei litere mici a alfabelului, să se modifice aceasta pentru a memora codul literei mari corespunzătoare.

Rezolvare

```
x = x - ('a' - 'A');
```

În loc de `'a'-'A'` puteam să scriem `'b'-'B'` sau `'p'-'P'` sau orice diferență între codurile unei perechi literă mică - literă mare identice. Acest lucru se datorează faptului că această diferență este constantă și trebuie să mai știm că literele mici au codurile mai mari decât cele mari. Această diferență este `32` dar noi nu trebuie deci să ținem minte acest lucru.

2. Având memorat în variabila `x` de tip `char` codul unei litere mici a alfabelului, să se afișeze pe ecran următoarea literă în alfabet (sau un mesaj dacă deja `x` memorează valoarea ultimei litere).

Rezolvare

```
if (x == 'z')
    cout<<"ultima";
else
    cout<<(char)(x+1);
```

3. Se citește de la tastatură o dată de tip `char`. Să se afișeze msajul `DA` dacă s-a introdus o literă mică a alfabetului sau mesajul `NU` dacă s-a introdus altceva.

Rezolvare

Literele mici ale alfabetul au codurile ASCII într-un interval, așadar testul este echivalent cu a compara doar cu marginile. Nu este necesară cunoașterea codurilor ASCII ale primei și ultimei litere, este suficient să scriem simbolul între apostrofuri pentru că am stabilit că operatorii se aplică oricum codului numeric.

```
#include <iostream>
```

```
using namespace std;
char x;
int main () {
    cin>>x;
    if (x >= 'a' && x <= 'z')
        cout<<"DA";
    else
        cout<<"NU";
    return 0;
}
```

4. Se citește de la tastatură o dată de tip `char`. Să se afișeze msajul *literă* dacă s-a introdus o literă a alfabetului (mare sau mică), mesajul *cifră* dacă s-a introdus un simbol cifră sau mesajul *altceva* în alt caz.

Rezolvare

```
#include <iostream>
using namespace std;
char x;
int main () {
    cin>>x;
    if ((x >= 'A' && x <= 'Z') || (x >= 'a' && x <= 'z'))
        cout<<"litera";
    else
        if (x >= '0' && x <= '9')
            cout<<"cifra";
        else
            cout<<"altceva";
    return 0;
}
```

Literele mari și cele mici reprezintă două intervale de coduri. Așadar testul de literă se reduce la a testa dacă ne aflăm în unul dintre aceste intervale. Știind că literele mari au codurile mai mici, poate suntem tentați să scriem `(x >= 'A' && x <= 'z')`. Acest lucru nu este corect întrucât cele două intervale nu sunt lipite. Fiind `26` de litere, `A` având codul `65`, obținem că `Z` are codul `90`. Deci de la `91` până la `96` sunt și alte caractere (`97` este codul lui `a`).

Trebuie să fim atenți la testul de cifră pentru că deseori se întâmplă să se uite apostrofurile. Nu se primește eroare de compilare dar nici nu se compară cu ce ne-am dori (codul ASCII al simbolului).

5. Se citește de la tastatură o dată de tip `char`. Să se verifice dacă reprezintă o vocală literă mică.

Rezolvare

```
#include <iostream>
using namespace std;
char x;
int main () {
    cin>>x;
    if (x=='a'||x=='e'||x=='i'||x=='o'||x=='u')
        cout<<"este vocala";
    else
        cout<<"nu este vocala";
```

```
    return 0;
}
```

Codurile ASCII ale vocalelor nu sunt consecutive așa că nu mai putem pune condiția de apartenență a caracterului la un interval. Fiind doar `5` vocale putem scrie o condiție compusă din alte `5` legate prin `||`. Dacă am avea de testat dacă este consoană caracterul citit nu trebuie să ne grăbim să considerăm că este corect doar să negăm condiția de mai sus. Asta ar însemna doar că nu este vorba despre o vocală. Trebuie să legăm cu `&&` negarea condiției de mai sus de condiția să fie literă mică a alfabetului caracterul de testat.

6. Să se afișeze pe ecran simbolurile (afișabile) din setul ASCII standard alături de codurile lor.

Rezolvare

Caracterele afișabile încep cu codul `32` (*spațiul*) și cele din setul standard le considerăm până la `127`. Cele cu coduri de la `128` la `255` reprezintă setul ASCII extins.

```
#include <iostream>
using namespace std;
int i;
int main () {
    for (int i=32;i<=127;i++)
        cout<<(char)i<<"
"<<i<<"\n";
    return 0;
}
```

```
 32    ! 33    " 34    # 35    $ 36    % 37    & 38    ' 39    ( 4
0    ) 41    * 42    + 43    , 44    - 45    . 46    / 47    0 48
 1 49    2 50    3 51    4 52    5 53    6 54    7 55    8 56    9
 57    : 58    ; 59    < 60    = 61    > 62    ? 63    @ 64    A 65
   B 66    C 67    D 68    E 69    F 70    G 71    H 72    I 73
 J 74    K 75    L 76    M 77    N 78    O 79    P 80    Q 81    R
82    S 83    T 84    U 85    V 86    W 87    X 88    Y 89    Z 90
   [ 91    \ 92    ] 93    ^ 94    _ 95    ` 96    a 97    b 98
c 99    d 100    e 101    f 102    g 103    h 104    i 105    j 106
    k 107    l 108    m 109    n 110    o 111    p 112    q 113
r 114    s 115    t 116    u 117    v 118    w 119    x 120    y 12
1    z 122    { 123    | 124    } 125    ~ 126    ⌂ 127
Process returned 0 (0x0)   execution time : 0.229 s
Press any key to continue.
```