Curs 14

Scrierea formatată a datelor în C++

1.Operatorii de inserţie/extracţie.

Operatorii de inserţie/extracţie , <</>>, pot fi utilizati numai pentru fişiere *text* şi numai pentru în operaţii de scriere/citire cu date de tip *numeric*, de tip *caracter* sau de tip *şir de caractere*. Aceste date au câte un anumit *format intern* şi pot avea mai multe *formate externe*. Formatul intern depinde atât de modalitatea de *reprezentare internă* cât şi de modul de stocare a datei în memorie, şi poate fi diferit de la o maşina la alta. De exemplu, pentru MS Visual C++ , un întreg de tip **signed int** este reprezentat pe 32 de biţi în complement faţă de doi, este scris în memorie pe 4 octeţi, primul fiind octetul cel mai puţin semnificativ. Formatele externe specifică modalitaţile de scriere a datelor cu ajutorul setului de caractere şi au rolul de a furniza utilizatorului, într-o formă inteligibilă, valoarea şi semnificaţia datelor reprezentate. Deoarece compilatorul trebuie să traducă în şi din format intern în format extern, regulile de alcatuire a formatelor externe sunt riguros definite în cadrul limbajului de programare. Formatul extern este folosit în cazul introducerii datelor de la tastatură sau în cazul afişării pe monitor. De asemenea, scrierea valorilor unor date direct în fişierul sursă al programului este supusă regulilor unui anumit format extern: formatul de scriere al constantelor (literali). In continuare vom reaminti aceste reguli.

Constantele de tip caracter se scriu între apostrofuri cu ajutorul caracterelor imprimabile sau al secvenţelor *escape*. Exemple: `'a', '\n', '\xa','\12'` (ultimele trei constante au aceeaşi valoare: caracterul "linie-nouă"). O variabilă de tip **char** poate fi interpretată în doua moduri distincte: ca fiind o variabilă de tip întreg (şi folosita ca atare în calcule aritmetice) sau ca fiind o variabilă care conţine codul unui caracter, având deci "tipul caracter". Intr-o expresie aritmetică, o variabilă de tip **char** este un numar întreg, iar intr-o operaţie de scriere/citire este un caracter. Formatul intern al unui caracter este dat de codul caracterului (MS Visual C++ foloseste codul ASCII cu o extindere specifică firmei Microsoft, OEM, tipul **char** este alocat pe un octet, dar pot fi folosite şi caractere pe doi octeti, pot fi folosite, de asemenea, şi alte moduri de codificare). Caracterele sunt elementele din care sunt alcatuite formatele externe ale celorlalte tipuri de date.

Constantele de tip şir de caractere (*string*) se scriu în codul sursă între ghilimele cu ajutorul caracterelor imprimabile sau al secventelor *escape*, exemple: `"abc", "\x61\x62\x63", "etc. \n"`. Acesta este formatul înţeles de către compilator. Formatul extern al unui şir de caractere nu foloseşte nici ghilimelele şi nici secvenţe *escape*. Formatul intern utilizează un octet nul pe post de terminator de şir.

Constantele întregi sunt scrise de obicei în baza zece (*scrire decimală*, cu prima cifră nenulă), dar pot fi scrise şi în baza opt (*scriere octală*) sau şaisprezece (*scriere hexazecimală*). Constantele octale sunt scrise cu prima cifră zero, constantele hexazecimale sunt prefixate cu **0x** sau cu **0X**. Orice constantă întreagă poate fi urmată de sufixul **u** sau **U** pentru **unsigned int** sau/şi de **l, ll,** sau **L, LL,** pentru **long int, long long int** (în absenţa sufixului au implicit tipul **int**). Exemplu: următoarele constante: `123, 0123` şi `0x123` au valori diferite, fiind scrise în baze diferite. Operatorul de citire >>

recunoaşte toate aceste formate. Operatorul de inserţie << poate să scrie (la cerere) prefixe pentru a indica baza de numeratie folosită, dar nu scrie şi sufixe.

Constantele flotante au în mod implicit tipul **double** şi admit două moduri de scriere: numai cu punct zecimal (*fixed*) sau cu mantisă şi exponent (*scientific*) (mantisa poate contine sau nu punctul zecimal). Orice constanta flotantă poate fi urmată de sufixele **f** sau **F** pentru **float** şi de **l** sau **L** pentru **long double**. Exemple: următoarele constante au aceeaşi valoare, dar tipuri diferite: `1; 1.0; 10e-1f; .1E1L` (prima are tipul **int**).

Limbajul nu defineşte "constante de tip adresă", adresele (valorile pointerilor) pot fi scrise totuşi într-un format specific, dar care depinde de implementare. în MS Visual C++ (cu setările implicite) adresele sunt reprezentate printr-un şir de opt cifre hexazecimale.

Analizam în continuare *comportamentul prestabilit* al operatorilor de inserţie şi de extracţie, <</>>, pentru a intelege cum îl putem modifica în cazul în care nu corespunde nevoilor noastre.

Operatorul de inserţie, <<, dacă are ca operand o variabilă, x, recunoaşte tipul variabilei şi are un comportament adecvat acestui tip (dacă x este de tip **char** va scrie un singur caracter, dacă x are tipul **char *** va scrie un şir de caractere, etc), iar dacă are de scris o constantă va recunoaşte atât tipul constantei cât şi valoarea ei. Operatorul << recunoaşte aşadar tipurile fundamentale şi întelege orice format extern de reprezentare a datelor, dar de scris va scrie numai în formatul prestabilit pentru acel tip de dată. Astfel, la scrierea unui caracter sau a unui şir de caractere va scrie codurile acestora, la scrierea unei date de tip întreg va folosi formatul *decimal* (în baza zece) fără semnul + pentru valori pozitive, iar la scrierea unei valori de tip flotant (**float**, **double** sau **long double**) va alege în mod automat, în funcţie de valoarea datei, formatul cel mai economic dintre *fixed* şi *scientific*. In toate cazurile, în fişierul de ieşire vor fi scrişi unul sau mai multi octeţi conţinând codurile caracterelor care formează formatul prestabilit al datei scrise, şi numai atât (nu se adaugă spaţii de separare, aliniere, etc.). Lăţimea *câmpului de editare* este variabilă şi are valoarea minimă necesară pentru scrierea datei.

Operatorul de extracţie, >>, se comportă şi el diferit în functie de tipul variabilei care va fi încărcată cu valoarea citită. La citirea unei valori numerice recunoaşte orice format de scriere, dar la citirea unui caracter sau a unui sir de caractere citeste direct codurile acestora , fară să recunoască şi constantele de aceste tipuri, cum ar fi `'a'` - scris cu tot cu apostrofuri- sau `"abc"` – scris cu ghilimele - . Mărimea *câmpului de citire* este variabilă şi se stabileşte după urmatoarele reguli: spaţiile albe iniţiale sunt sărite, citirea valorii începe de la primul caracter care nu este spaţiu alb (blanc, tab, linie-nouă sau retur-car) şi continuă până la primul separator sau până la primul caracter care nu are nici o semnificaţie în formatul citit. Separatorii sunt spaţiile albe. Pentru citirea valorilor numerice acest comportament este mulţumitor, dar pentru caractere sau şiruri de caractere are destule neajunsuri: nu pot fi citite caracterele cu rol de separator, nu pot fi citite stringuri care conţin blanc-uri, etc.

2. Funcţii de formatare

Comportamentul prestabilit al operatorilor de inserţie/extracţie este acelaşi pentru toate stream-urile I/O şi este dat de *valorile implicite* ale unor variabile de tip întreg sau de tip "indicator pe un bit", variabile care precizează anumite *caracteristici de formatare*, cum ar fi: lăţimea minimă a câmpului de editare, formatul de scriere al valorilor numerice, numărul de cifre semnificative, etc. Aceste atribute sunt moştenite de la clasa **ios** şi pot fi modificate prin intermediul următoarelor metode:

`int width ();`	Returnează lăţimea curentă a campului de editare;
`int width (int);`	Stabileşte nouă laţime, o întoarce pe cea veche;
`char fill ();`	Returnează caracterul de umplere curent ;
`char fill (char);`	Stabileşte noul caracter de umplere, îl întoarce pe cel vechi;
`int precision ();`	Returnează precizia curentă;
`int precision (int);`	Stabilieşte noua precizie, o returnează pe cea veche;
`long setf (long);`	Setează numai indicatorii de formatare dati de parametru;
`long setf (long, long);`	Sterge câmpul de formatare dat de al doilea parametru şi setează numai indicatorii daţi de primul parametru.
`long unsetf (long);`	Sterge indicatorii de formatare daţi de parametru.
`long flags (long);`	Steaza toti indicatorii, cei care nu apar în parametru sunt sterşi (flags(0) restabileste formatul implicit);
`long flags (void);`	Toate funcţiile de formatare returnează valoarea precedentă a vectorului de formatare.

Caracteristicile de formatare afectează numai operatorii de inserţie/extracţie, nu şi celelate funcţii de scriere/citire, cum ar fi `put()/get()` sau `write()/read()`.

Lăţimea câmpului de editare. La scrierea unei date poate fi precizată lăţimea minimă a câmpului de editare (dar nu şi cea maximă). Lăţimea minimă implicită este zero şi poate fi schimbata cu **width()**. Noua valoare este valabilă numai pentru operaţia de scriere imediat următoare (şi numai dacă se foloseşte operatorul de inserţie!), după care se revine automat la valoarea implicită. Acest comportament este specific lăţimii câmpului de editare, asupra celorlalte caracteristici de formatare modificarile au un caracter persistent, revenirea la valorile prestabilite este posibilă, dar trebuie cerută în mod explicit.

Caracterul de completare. Dacă la scrierea unei date nu este atinsă lăţimea minimă cerută a câmpului de editare, acesta este completat, în mod implicit, cu blancuri. Caracterul de completare poate fi schimbat cu functia **fill()**.

Exemplu:

```
#include<iostream>
using namespace std;
int main()
{
    double x = 12.3456;
    double xx = 1234567.3456;
    cout.width(12);
    cout.fill('#');
    cout << "test1" << "---"<< "test2" << endl;
    cout.width(12);
    cout <<"x="<< x << endl;
    cout.width(12);
    cout << "x=";
    cout.width(12);
    cout << x << endl;
    cout.width(12);
    cout << "mai-mult-de-12-caractere" << endl;
    return 0;
}

//#######test1---test2
//###########x=12.3456
//###########x=#####12.3456
//mai - mult - de - 12 - caractere
//Press any key to continue . . .
```

Precizia. In mod implicit, valorile numerice flotante (**float** sau **double**) sunt imprimate cât mai compact posibil, alegându-se în mod automat fie formatul cu punct zecimal, fie cel "ştiinţific" (mantisa şi exponent), şi utilizandu-se cel mult 6 cifre semnificative. Acest număr maxim de cifre utilizate poate fi shimbat cu funcţia **precision().** In cazul în care utilizatorul a impus (cu functia **setf()**, de exemplu) utilizarea unui anumit format de scriere (*fixed* sau *scientific*), funcţia **precision()** stabileşte numărul de cifre care apar după punctul zecimal.

```
#include<iostream>
using namespace std;
int main()
{
    double x = 123.456789;
    double xx = 1234567.89;
    cout << "\n IMPLICIT:" << endl;
    cout << x << endl;
    cout << xx << endl;
    cout << "\n PRECIZIA 12" << endl;
    cout.precision(12);
    cout << x << endl;
    cout << xx << endl;
    cout << "\n FIXED:" << endl;
    cout.setf(ios::fixed, ios::floatfield);
    cout << x << endl;
    cout << xx << endl;
    return 0;
}
```

```
//IMPLICIT:
//123.457
//1.23457e+06
//
//PRECIZIA 12
//123.456789
//1234567.89
//
//FIXED:
//123.456789000000
//1234567.889999999898
//Press any key to continue . . .
```

Funcţiile de formatare. Functiile **setf()**/**unsetf()** şi **flags()** modifică la nivel de bit valoarea "vectorului" de formatare (care este de fapt o variabila de tip **long**), utilizând urmatoarele constante declarate în clasa **ios**:

Indicator de formatare	Câmp de formatare	I/O	Semnificatie
`ios::left`	`ios::adjustfield`	out	Aliniere la stânga (*left-adjust*)
`ios::right`	`ios::adjustfield`	out	Aliniere la dreapta (*right-adjust*)
`ios::dec`	`ios::basefield`	i/o	Scriere în baza 10 (*decimal*).
`ios::oct`	`ios::basefield`	i/o	Scriere în baza 8 (*octal*).
`ios::hex`	`ios::basefield`	i/o	Scriere în baza 16 (*hexadecimal*).
`ios::fixed`	`ios::floatfield`	i/o	Scriere cu punct zecimal.
`ios::scientific`	`ios::floatfield`	i/o	Scriere cu mantisa şi exponent.
`ios::uppercase`	-	out	A,B,…F pentru cifre hexazecimale.
`ios::showbase`	-	out	Scriere prefixata pentru constante întregi
`ios::showpoint`	-	out	Impune scrierea punctului zecimal.
`ios::showpos`	-	out	Impune scrierea lui + pentru nr. pozitive.
`ios::skipws`	-	in	Salt peste blancuri. (*Skip white spaces*)

Exemplificari.

Dacă dorim ca operatorul >> să poata citi orice caracter (inclusiv '\n') utilizăm apelul: `unsetf(ios::skipws)`. Atenţie, în acest caz este afectat numai modul predefinit de citire al variabilelor de tip caracter, nu şi al stringurilor.

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

int main(){
    char ch;
    cout << "Pentru STOP tastati S" << endl;
    cin.unsetf(ios::skipws);
    do{
        cin >> ch;
        cout << ch;
    } while (ch != 'S');
    cout << "TOP" << endl;

    return 0;
}
/* MONITOR:
Pentru STOP tastati S
abc def
abc def
xySzzzz zzzz
xySTOP
Press any key to continue . . .
*/
```

Revenirea la comportamentul implicit: `setf(ios::skipws)`.

Apariţia semnului + pentru valori pozitive şi a puncul zecimal pentru valori întregi ale unei date flotante se poate cere astfel:

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

int main()
{
    double db = 10;
    cout << "db=" << db << endl;
    cout.setf(ios::showpos | ios::showpoint);
    cout << "db=" << db << endl;
    return 0;
}
/* MONITOR:

db=10
db=+10.0000
Press any key to continue
*/
```

Indicatorii `ios::showbase` şi `ios::uppercase` se folosesc doar în cazul în care am schimbat baza de numeraţie:

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

int main()
```

```
{
    int i = 161;
    cout.setf(ios::hex, ios::basefield);
    cout << "i=" << i << endl;
    cout.setf(ios::showbase | ios::uppercase);
    cout << "i=" << i << endl;
    return 0;
}
/* MONITOR:

i=a1
i=0XA1
Press any key to continue

*/
```

Pentru a impune un anumit format la editarea valorilor numerice trebuie folosită forma cu doi parametri a funcţiei **setf().** Prezenţa câmpului de formatare este necesară, mai întâi sunt sterşi toţi indicatorii câmpului şi apoi este setat numai cel specificat de primul parametru, altfel rămâne setat şi vechiul indicator (caz în care stream-ul se va comporta în modul prestabilit).

```
#include<iostream>
using namespace std;
int main()
{
    int i = 64;
    cout << "i=" << i << endl;
    cout.setf(ios::oct, ios::basefield);
    cout << "i=" << i << endl;
    cout.setf(ios::showbase);
    cout << "i=" << i << endl << endl;
    double db = 13.12345678901234;
    cout << "db=" << db << endl;
    cout.setf(ios::scientific, ios::floatfield);
    cout << "db=" << db << endl;
    cout.precision(12);
    cout << "db=" << db << endl << endl;

    return 0;
}
//i = 64
//i = 100
//i = 0100
//
//db = 13.1235
//db = 1.312346e+001
//db = 1.312345678901e+001
//
//Press any key to continue . . .
```

Indicatorii prezentaţi mai sus pot fi modificaţi şi pentru stream-uri de intrare, impunând astfel o anumită interpretare a câmpurilor citite. Atenţie: operatorul >> va recunoaşte numai formatul specificat:

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

int main()
{
    int i;
    cin.setf(ios::oct, ios::basefield);
    cin >> i;
    cout << "i=" << i << endl;;
    return 0;
}
/* MONITOR 1:
       10
       i=8
       Press any key to continue
MONITOR 2:
       0XAA
       i=0
       Press any key to continue*/
```

In cazul în care lăţimea minimă a câmpului de editare este setată la o valoare strict mai mare decat numarul de caractere care urmează să fie scrise, acestea sunt aliniate în mod implicit la dreapta câmpului de editare. Alinierea la stanga se obţine astfel:

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

int main()
{
    int i = 64;

    cout << "^^^";
    cout.width(12);
    cout << i << "^^^" << endl;

    cout.setf(ios::left, ios::adjustfield);
    cout << "^^^";
    cout.width(12);
    cout << i << "^^^" << endl;

    return 0;
}

/* MONITOR:
^^^          64^^^
^^^64          ^^^
Press any key to continue*/
```

Un stil corect de programare presupune salvarea caracteristicilor de formatare înaintea oricarei modificări ale acestora şi revenirea la starea lor iniţiala înainte de a părăsi corpul funcţiei în care s-au efectuat modificările. Exemplu:

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

```
void afisare(const char **ppc, int dim)
{

    long stareaInitiala = cout.flags();
    long stangaDreapta;
    for (int i = 0; i<dim; i++) {
        stangaDreapta = (i % 2 ? ios::right : ios::left);
        cout.setf(stangaDreapta, ios::adjustfield);
        cout.width(15);
        cout << *ppc++ << endl;
    }
    cout.flags(stareaInitiala);
    return;
}
int main(void)
{
    const char* sTab[] = { "Inca","un","exemplu","mai","mult",
        "decat","edificator:","A!","\n" };
    afisare(sTab, 9);
    cout << "Clar, nu?" << endl;
    return 0;
}
```

Observaţie: la setările prestabilite se poate reveni cu apelul `cout.flags(0);`.

3. Manipulatori de formatare.

Un *manipulator* este un obiect care poate fi inserat (sau extras) într-un stream cu scopul de a produce un anumit efect. Utilizarea unui anumit manipulator este echivalentă cu o anumită secvenţă de apeluri către funcţii membre ale stream-ului. Un exemplu uzual este manipulatorul **endl,** care inserat într-un stream de ieşire provoacă trecerea la o linie nouă urmată de golirea buffer-ului (adică transferul caracterelor din buffer în fişierul asociat). Secvenţa:

```
cout << "test" << endl;
```

este echivalenta cu:

```
cout << "test" << '\n';
cout.flush();
```

Majoritatea operaţiilor de formatare pot fi efectuate mult mai uşor prin folosirea unor manipulatori predefiniţi decât prin utilizarea directă a funcţiilor de formatare. De exemplu,

```
cout << oct << 64 << endl;
```

reprezintă o cale mai comoda de a obţine efectul dat de secvenţa de cod:

```
cout.setf(ios::oct, ios::basefield);
cout << 64 << endl;
```

Unii manipulatori admit parametri, aceşti manipulatori, pentru a fi recunoscuţi de compilator, necesită includerea şi a fişierului header <**iomanip**>.

Manipulator	I/O	Efect
`endl`	out	Inserează linie-nouă şi goleşte buffer-ul.
`ends`	out	Inserează terminatorul de string.
`flush`	out	Goleşte bufferul de ieşire.
`dec`	i/o	`setf( ios::dec, ios::basefield );`
`hex`	i/o	`setf( ios::hex, ios::basefield );`
`oct`	i/o	`setf( ios::oct, ios::basefield );`
`ws`	in	`setf( ios::skipws );`
`setbase( int )`	i/o	Seteaza baza de numeratie la 8, 10, sau 16.
`setiosflags( long )`	i/o	`setf( long ) sau setf( long, long );`
`resetiosflags( long )`	i/o	`unsetf( long );`
`setfill( int )`	i/o	`fill( int );`
`setprecision( int )`	i/o	`precision( int );`
`setw( int )`	i/o	`width(int )`

□

Exemple de utilizare:

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


int main()
{
    int i = 64;
    cout << hex << i << endl;
    cout << setfill('*') << setw(12) << i << '\n' << endl;

    cout << dec << i << endl;
    cout << setw(12) << i << '\n' << endl;

    cout << setiosflags(ios::left) << setw(12) << i << endl;
    cout << resetiosflags(ios::left) << setw(12) << i << endl;

    return 0;
}
/* MONITOR:
40
**********40

64
**********64

64**********
**********64
Press any key to continue

*/
```

Observaţie: toate exemplele prezentate aici au folosit, din motive de comoditate, numai stream-urile predefinite **cin** şi **cout**. Modificarea caracteristicilor de formatare se face în

acelaşi mod pentru orice stream din clasele **istream**, **ostream**, **iostream**, **ifstream**, **ofstream** sau **fstream**.

```
/* Acest program scrie codurile ASCII în fişierul D:\ascii.txt  */

#include<iostream>
#include<fstream>
#include<iomanip>
using namespace std;
int main(){
    int i, j, icod;
    char numeFisier[] = "D:\\ascii.txt";
    ofstream xout;
    xout.open(numeFisier);
    if (!xout.good()) {
        cout << " Eroare: vezi ce se intampla cu " << numeFisier;
        return 0;
    }
    xout << "\t\tCoduri ASCII\n" << endl;
    xout << "\t (caractere imprimabile)\n" << endl;
    for (j = 0; j<4; j++)
        xout << "   DEC  HEX";
    xout << endl;
    xout.setf(ios::uppercase);
    for (i = 32; i<88; i++) {
        xout << " I";
        for (j = 0; j<4; j++) {
            icod = i + j * 56;
            xout << dec << setw(4) << icod << setw(2) << (char)icod;
            xout << hex << setw(3) << icod;
            xout << " I";
        }
        xout << endl;
    }
    xout.close();
    return 0;
}
```

REZULTAT: Fişierul ascii.txt deschis cu Microsoft Word, codificare MS-DOS OEM United States

```
                Coduri ASCII

            (caractere imprimabile)

   DEC  HEX    DEC  HEX    DEC  HEX    DEC  HEX
 I  32    20 I  88 X 58 I 144 É 90 I 200 ╚ C8 I
 I  33 ! 21 I  89 Y 59 I 145 æ 91 I 201 ╔ C9 I
 I  34 " 22 I  90 Z 5A I 146 Æ 92 I 202 ╩ CA I
 I  35 # 23 I  91 [ 5B I 147 ô 93 I 203 ╦ CB I
 I  36 $ 24 I  92 \ 5C I 148 ö 94 I 204 ╠ CC I
```

I	37	%	25	I	93	]	5D	I	149	ò	95	I	205	═	CD	I
I	38	&	26	I	94	^	5E	I	150	û	96	I	206	╬	CE	I
I	39	'	27	I	95	_	5F	I	151	ù	97	I	207	╧	CF	I
I	40	(	28	I	96	`	60	I	152	ÿ	98	I	208	╨	D0	I
I	41	)	29	I	97	a	61	I	153	Ö	99	I	209	╤	D1	I
I	42	*	2A	I	98	b	62	I	154	Ü	9A	I	210	╥	D2	I
I	43	+	2B	I	99	c	63	I	155	¢	9B	I	211	╙	D3	I
I	44	,	2C	I	100	d	64	I	156	£	9C	I	212	╘	D4	I
I	45	-	2D	I	101	e	65	I	157	¥	9D	I	213	╒	D5	I
I	46	.	2E	I	102	f	66	I	158	₧	9E	I	214	╓	D6	I
I	47	/	2F	I	103	g	67	I	159	ƒ	9F	I	215	╫	D7	I
I	48	0	30	I	104	h	68	I	160	á	A0	I	216	╪	D8	I
I	49	1	31	I	105	i	69	I	161	í	A1	I	217	┘	D9	I
I	50	2	32	I	106	j	6A	I	162	ó	A2	I	218	┌	DA	I
I	51	3	33	I	107	k	6B	I	163	ú	A3	I	219	█	DB	I
I	52	4	34	I	108	l	6C	I	164	ñ	A4	I	220	▄	DC	I
I	53	5	35	I	109	m	6D	I	165	Ñ	A5	I	221	▌	DD	I
I	54	6	36	I	110	n	6E	I	166	ª	A6	I	222	▐	DE	I
I	55	7	37	I	111	o	6F	I	167	º	A7	I	223	▀	DF	I
I	56	8	38	I	112	p	70	I	168	¿	A8	I	224	α	E0	I
I	57	9	39	I	113	q	71	I	169	⌐	A9	I	225	ß	E1	I
I	58	:	3A	I	114	r	72	I	170	¬	AA	I	226	Γ	E2	I
I	59	;	3B	I	115	s	73	I	171	½	AB	I	227	π	E3	I
I	60	<	3C	I	116	t	74	I	172	¼	AC	I	228	Σ	E4	I
I	61	=	3D	I	117	u	75	I	173	¡	AD	I	229	σ	E5	I
I	62	>	3E	I	118	v	76	I	174	«	AE	I	230	µ	E6	I
I	63	?	3F	I	119	w	77	I	175	»	AF	I	231	τ	E7	I
I	64	@	40	I	120	x	78	I	176	░	B0	I	232	Φ	E8	I
I	65	A	41	I	121	y	79	I	177	▒	B1	I	233	Θ	E9	I
I	66	B	42	I	122	z	7A	I	178	▓	B2	I	234	Ω	EA	I
I	67	C	43	I	123	{	7B	I	179	│	B3	I	235	δ	EB	I
I	68	D	44	I	124	|	7C	I	180	┤	B4	I	236	∞	EC	I
I	69	E	45	I	125	}	7D	I	181	╡	B5	I	237	φ	ED	I
I	70	F	46	I	126	~	7E	I	182	╢	B6	I	238	ε	EE	I
I	71	G	47	I	127		7F	I	183	╖	B7	I	239	∩	EF	I
I	72	H	48	I	128	Ç	80	I	184	╕	B8	I	240	≡	F0	I
I	73	I	49	I	129	ü	81	I	185	╣	B9	I	241	±	F1	I
I	74	J	4A	I	130	é	82	I	186	║	BA	I	242	≥	F2	I
I	75	K	4B	I	131	â	83	I	187	╗	BB	I	243	≤	F3	I
I	76	L	4C	I	132	ä	84	I	188	╝	BC	I	244	⌠	F4	I
I	77	M	4D	I	133	à	85	I	189	╜	BD	I	245	⌡	F5	I
I	78	N	4E	I	134	å	86	I	190	╛	BE	I	246	÷	F6	I
I	79	O	4F	I	135	ç	87	I	191	┐	BF	I	247	≈	F7	I
I	80	P	50	I	136	ê	88	I	192	└	C0	I	248	°	F8	I
I	81	Q	51	I	137	ë	89	I	193	┴	C1	I	249	∙	F9	I
I	82	R	52	I	138	è	8A	I	194	┬	C2	I	250	·	FA	I
I	83	S	53	I	139	ï	8B	I	195	├	C3	I	251	√	FB	I
I	84	T	54	I	140	î	8C	I	196	─	C4	I	252	ⁿ	FC	I
I	85	U	55	I	141	ì	8D	I	197	┼	C5	I	253	²	FD	I
I	86	V	56	I	142	Ä	8E	I	198	╞	C6	I	254	■	FE	I
I	87	W	57	I	143	Å	8F	I	199	╟	C7	I	255		FF	I