

# Scrierea formatată a datelor în C++

## 1.Operatorii de inserție/extracție.

Operatorii de inserție/extracție, `<</>>`, pot fi utilizați numai pentru fișiere *text* și numai pentru în operații de scriere/citire cu date de tip *numeric*, de tip *caracter* sau de tip *șir de caractere*. Aceste date au câte un anumit *format intern* și pot avea mai multe *formate externe*. Formatul intern depinde atât de modalitatea de reprezentare internă cât și de modul de stocare a datei în memorie, și poate fi diferit de la o mașină la alta. De exemplu, pentru MS Visual C++, un întreg de tip **signed int** este reprezentat pe 32 de biți în complement față de doi, este scris în memorie pe 4 octeți, primul fiind octetul cel mai puțin semnificativ. Formatele externe specifică modalitățile de scriere a datelor cu ajutorul setului de caractere și au rolul de a furniza utilizatorului, într-o formă inteligibilă, valoarea și semnificația datelor reprezentate. Deoarece compilatorul trebuie să traducă în și 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 afișării pe monitor. De asemenea, scrierea valorilor unor date direct în fișierul sursă al programului este supusă regulilor unui anumit format extern: formatul de scriere al constantelor (literali). În continuare vom reaminti aceste reguli.

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

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

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

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

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

Limbajul nu definește “constante de tip adresă”, adresele (valorile pointerilor) pot fi scrise totuși într-un format specific, dar care depinde de implementare. În MS Visual C++ (cu setările implicite) adresele sunt reprezentate printr-un șir de opt cifre hexazecimale.

Analizăm în continuare *comportamentul prestabilit* al operatorilor de inserție și de extracție, <</>>, pentru a înțelege cum îl putem modifica în cazul în care nu corespunde nevoilor noastre.

Operatorul de inserție, <<, dacă are ca operand o variabilă, x, recunoaște tipul variabilei și 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 șir de caractere, etc), iar dacă are de scris o constantă va recunoaște atât tipul constantei cât și valoarea ei. Operatorul << recunoaște așadar tipurile fundamentale și înțelege 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 șir 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 funcție de valoarea datei, formatul cel mai economic dintre *fixed* și *scientific*. În toate cazurile, în fișierul de ieșire vor fi scriși unul sau mai mulți octeți conținând codurile caracterelor care formează formatul prestabilit al datei scrise, și numai atât (nu se adaugă spații de separare, aliniere, etc.). Lățimea *câmpului de editare* este variabilă și are valoarea minimă necesară pentru scrierea datei.

Operatorul de extracție, >>, se comportă și el diferit în funcție de tipul variabilei care va fi încărcată cu valoarea citită. La citirea unei valori numerice recunoaște orice format de scriere, dar la citirea unui caracter sau a unui șir de caractere citește direct codurile acestora, fără să recunoască și constantele de aceste tipuri, cum ar fi 'a' - scris cu tot cu apostrofuri- sau "abc" - scris cu ghilimele - . Mărimea *câmpului de citire* este variabilă și se stabilește după următoarele reguli: spațiile albe inițiale sunt sărite, citirea valorii începe de la primul caracter care nu este spațiu alb (blanc, tab, linie-nouă sau retur-car) și continuă până la primul separator sau până la primul caracter care nu are nici o semnificație în formatul citit. Separatorii sunt spațiile albe. Pentru citirea valorilor numerice acest comportament este mulțumitor, dar pentru caractere sau șiruri de caractere are destule neajunsuri: nu pot fi citite caracterele cu rol de separator, nu pot fi citite stringuri care conțin blanc-uri, etc.

## 2. Funcții de formatare

Comportamentul prestabilit al operatorilor de inserție/extracție este același pentru toate stream-urile I/O și 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: lățimea minimă a câmpului de editare, formatul de scriere al valorilor numerice, numărul de cifre semnificative, etc. Aceste atribute sunt moștenite de la clasa **ios** și pot fi modificate prin intermediul următoarelor metode:

<code>int width ();</code>	Returnează lățimea curentă a câmpului de editare;
<code>int width (int);</code>	Stabilește nouă lățime, o întoarce pe cea veche;
<code>char fill ();</code>	Returnează caracterul de umplere curent ;
<code>char fill (char);</code>	Stabilește noul caracter de umplere, îl întoarce pe cel vechi;
<code>int precision ();</code>	Returnează precizia curentă;
<code>int precision (int);</code>	Stabilește noua precizie, o returnează pe cea veche;
<code>long setf (long);</code>	Setează numai indicatorii de formatare dați de parametru;
<code>long setf (long, long);</code>	Sterge câmpul de formatare dat de al doilea parametru și setează numai indicatorii dați de primul parametru.
<code>long unsetf (long);</code>	Sterge indicatorii de formatare dați de parametru.
<code>long flags (long);</code>	Șterge toți indicatorii, cei care nu apar în parametru sunt sterși ( <code>flags(0)</code> restabilește formatul implicit);
<code>long flags (void);</code>	Toate funcțiile de formatare returnează valoarea precedentă a vectorului de formatare.

Caracteristicile de formatare afectează numai operatorii de inserție/extracție, nu și celelate funcții de scriere/citire, cum ar fi `put()/get()` sau `write()/read()`.

Lățimea câmpului de editare. La scrierea unei date poate fi precizată lățimea minimă a câmpului de editare (dar nu și cea maximă). Lățimea minimă implicită este zero și poate fi schimbată cu **width()**. Noua valoare este valabilă numai pentru operația de scriere imediat următoare (și numai dacă se folosește operatorul de inserție!), după care se revine automat la valoarea implicită. Acest comportament este specific lățimii câmpului de editare, asupra celorlalte caracteristici de formatare modificările 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 lățimea minimă cerută a câmpului de editare, acesta este completat, în mod implicit, cu blăncuri. Caracterul de completare poate fi schimbat cu funcția **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. În mod implicit, valorile numerice flotante (**float** sau **double**) sunt imprimare cât mai compact posibil, alegându-se în mod automat fie formatul cu punct zecimal, fie cel “științific” (mantisa și exponent), și utilizându-se cel mult 6 cifre semnificative. Acest număr maxim de cifre utilizate poate fi schimbat cu funcția **precision()**. În cazul în care utilizatorul a impus (cu funcția **setf()**, de exemplu) utilizarea unui anumit format de scriere (*fixed* sau *scientific*), funcția **precision()** stabilește 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 . . .

```

Funcțiile de formatare. Funcțiile **setf()**/**unsetf()** și **flags()** modifică la nivel de bit valoarea “vectorului” de formatare (care este de fapt o variabila de tip **long**), utilizând următoarele constante declarate în clasa **ios**:

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

### Exemplificari.

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

Apariția semnelui + pentru valori pozitive și a punctul 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` și `ios::uppercase` se folosesc doar în cazul în care am schimbat baza de numerație:

```

#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 funcției **setf()**. Prezența câmpului de formatare este necesară, mai întâi sunt sterși toți indicatorii câmpului și apoi este setat numai cel specificat de primul parametru, altfel rămâne setat și 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 prezentați mai sus pot fi modificați și pentru stream-uri de intrare, impunând astfel o anumită interpretare a câmpurilor citite. Atenție: operatorul >> va recunoaște 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*/

```

În cazul în care lățimea minimă a câmpului de editare este setată la o valoare strict mai mare decât numărul de caractere care urmează să fie scrise, acestea sunt aliniate în mod implicit la dreapta câmpului de editare. Alinierea la stanga se obține 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 oricărei modificări ale acestora și revenirea la starea lor inițială înainte de a părăsi corpul funcției în care s-au efectuat modificările. Exemplu:

```

#include<iostream>
using namespace std;

```



```

void afisare(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)
{
    char *sTab[] = { "Inca", "un", "exemplu", "mai", "mult",
                    "decat", "edificator:", "A!", "\n" };
    afisare(sTab, 9);
    cout << "Clar, nu?" << endl;
    return 0;
}

```

Observație: 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 secvență de apeluri către funcții membre ale stream-ului. Un exemplu uzual este manipulatorul **endl**, care inserat într-un stream de ieșire provoacă trecerea la o linie nouă urmată de golirea buffer-ului (adică transferul caracterelor din buffer în fișierul asociat). Secvența:

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

este echivalenta cu:

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

Majoritatea operațiilor de formatare pot fi efectuate mult mai ușor prin folosirea unor manipulatori predefiniți decât prin utilizarea directă a funcțiilor de formatare. De exemplu,

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

reprezintă o cale mai comoda de a obține efectul dat de secvența de cod:

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

Unii manipulatori admit parametri, acești manipulatori, pentru a fi recunoscuți de compilator, necesită includerea și a fișierului header `<iomanip>`.

Manipulator	I/O	Efect
endl	out	Inserează linie-nouă și golește buffer-ul.
ends	out	Inserează terminatorul de string.
flush	out	Golește bufferul de ieșire.
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	Setează 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
*/
```

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

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

```

/* Acest program scrie codurile ASCII în fișierul 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: Fișierul 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	33	!	I	89	Y 59	I	145	æ 91
I	34	"	I	90	Z 5A	I	146	Æ 92
I	35	#	I	91	[ 5B	I	147	ô 93
I	36	\$	I	92	\ 5C	I	148	ö 94
						I	200	ℒ C8
						I	201	ℒ C9
						I	202	ℒ CA
						I	203	ℒ CB
						I	204	ℒ CC

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	ç	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	f	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	ç	A8	I	224	α	E0	I
I	57	9	39	I	113	q 71	I	169	ƒ	A9	I	225	β	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	I	184	☼	B8	I	240	≡	F0	I
I	73	I	49	I	129	ü	I	185	☼	B9	I	241	±	F1	I
I	74	J	4A	I	130	é	I	186	☼	BA	I	242	≥	F2	I
I	75	K	4B	I	131	â	I	187	☼	BB	I	243	≤	F3	I
I	76	L	4C	I	132	ä	I	188	☼	BC	I	244	∫	F4	I
I	77	M	4D	I	133	à	I	189	☼	BD	I	245	∫	F5	I
I	78	N	4E	I	134	å	I	190	☼	BE	I	246	÷	F6	I
I	79	O	4F	I	135	ç	I	191	☼	BF	I	247	≈	F7	I
I	80	P	50	I	136	è	I	192	☼	C0	I	248	°	F8	I
I	81	Q	51	I	137	ë	I	193	☼	C1	I	249	·	F9	I
I	82	R	52	I	138	è	I	194	☼	C2	I	250	·	FA	I
I	83	S	53	I	139	ï	I	195	☼	C3	I	251	√	FB	I
I	84	T	54	I	140	î	I	196	☼	C4	I	252	ⁿ	FC	I
I	85	U	55	I	141	ì	I	197	☼	C5	I	253	²	FD	I
I	86	V	56	I	142	Ä	I	198	☼	C6	I	254	■	FE	I
I	87	W	57	I	143	Å	I	199	☼	C7	I	255		FF	I