Analiza temei

Înaintea rezolvării unei teme se analizează evident cerinţele şi se revăd cunoştinţele necesare rezolvării. Să analizăm deci punct cu punct cerinţele temei propuse.

1. *Citirea unui cod format din maximum 9 cifre şi a unei denumiri de produs (şir de caractere). Dacă s-a citit un cod mai scurt de 9 caractere se va completa codul citit până la nouă caractere prin adăugarea în faţa secvenţei citite a unor cifre '7'.*
2. *Calculul cifrei de control pentru un cod complet, de 12 cifre. Codul complet se obţina adăugând în faţa celor nouă obţinute la punctul 1 a încă trei cifre care vor fi sistematic 594 (codul oficial al României).*

Deci ni se cere să citim de la tastatură un *şir de caractere* având maximum nouă caractere, toate cifre. Citirea unui şir de caractere apare în cursul 2:

```
char s[200];
int i;
cout  << "Introduceti un sir, fara spatii:";
cin >> s;
```

În program ar trebui deci să declarăm pentru şirurile citite de la tastatură (codul şi denumirea produsului) variabile de tip şir de caractere:

```
char codDat[10], denumire[21];
```

Pentru citirea celor două şiruri procedăm ca în curs:

```
cout << "Introduceti denumirea produsului (max. 20 caractere): ";
cin >> denumire;
cout << "Introduceti codul produsului (max. 9 cifre, fara spatii): ";
cin >> codDat;
```

Observaţie: La declararea lungimii unui şir de caractere se ţine cont de caracterul suplimentar '\0' (adăugat automat de *cout* la sfârşit), deci lungimile din declaraţii sunt bine scrise.

În continuare trebuie să completăm codul dat cu caractere '7', dacă lungimea citită e mai mică decât 9. Se poate adăuga însă şi o secvenţă de verificare a caracterelor, care trebuie să fie toate cifre. Ambele sarcini presupun cunoaşterea numărului de caractere din şirul *codDat*, determinat prin apelul funcţiei *strlen()* (cursul 7):

```
int lungime = strlen(codDat);
//  Verificare. Numar toate caracterele bune folosind un contor.
int i = 0;
while(codDat[i] >= '0' && codDat[i] <= '9')
      i++;
if(i != lungime)
{
      cout << "Cod eronat: " << codDat << endl;
      return;
}
```

Dacă s-a trecut de secvenţa de validare se poate genera şirul de caractere '7' necesar formării codului.

```
char sirSapte[9] = "";  //  sirul initial este vid
if(lungime < 9)
{
      int j;
      for(j=0; j<9-lungime; j++)
            sirSapte[j] = '7';
      //  Limitez sirSapte la dreapta adaugandu-i un caracter '\0'
      sirSapte[j] = '\0';
}
```

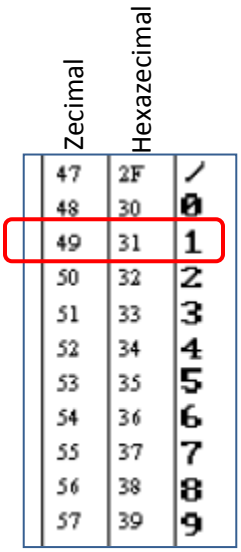
Avem în acest moment două şiruri, *sirSapte* şi *sirDat*. Ele ar trebui unite, dar codul final, de 13 cifre, are în faţă şirul "594", codul României. Şirul "594" poate fi şi el declarat, dar fiind constant poate fi şi iniţializat:

```
char codRO[] = "594";
```

Acum poate fi apelată repetat funcţia de unire a două şiruri (*strcat()*), ca în cursul 7:

```
char ean13[13];  //  13 caractere, 12+1
strcpy(ean13, codRO);
strcat(ean13, sirSapte);
strcat(ean13, codDat);
//  Verific pana aici
cout << "Primele 12 car. sunt : " << ean13 << endl;
```

Cele 12 caractere afişate nu pot fi din păcate folosite mai departe, pentru calculul caracterului de control (caracterul al 13-lea). Pentru a continua prelucrările cerute, cele 12 caractere obţinute trebuie să devină valori numerice. Dacă examinaţi tabela ASCII (cursul 2), caracterul '1' are codul zecimal 49. Interpretat ca număr este deci 49! Neplăcut.



Din fericire cifrele de la 0 la 9 apar în tabela ASCII una după alta. Şi mai există un lucru important, respectiv faptul că în C (C++) tipul *char* înseamnă în acelaşi timp caracter dar şi un întreg mic, pe 8 biţi, deci poate fi folosit la calcule, ca orice număr. Dacă am scădea din caracterele obţinute valoarea corespunzând caracterului '0' care este 48 (în tabela ASCII, desigur), am obţine valorile întregi necesare. Exemplu:

caracterul '7' are codul 55;

55-'0' = 55 – 48 = 7.

Excelent! Putem proceda aşa cu toate cele 12 caractere. În plus, după fiecare scădere rezultatul poate fi atribuit unei variabile întregi.

```
//  Convertim caracterele in valori numerice intregi (int)
int nean13[13];  //  Pentru rezultate
for(i=0; i<12; i++)
      nean13[i] = ean13[i] - '0';
```

Cele 12 valori întregi pot fi afişate pentru a verifica algoritmul.

Calculul cifrei de control

Calculula cifrei de control este prezentat în pagina web recomandată ca bibliografie (http://en.wikipedia.org/wiki/International_Article_Number_(EAN)) .

Practic va trebui să calculăm două sume, respectiv suma numerelor din poziţiile impare ale şirului de 12 valori obţinut (1, 3, ...) notată în continuare cu cu *s1* şi suma numerelor din

poziţiile pare (2, 4, ...) notată cu *s2*. Apoi se va calcula o valoare *S* în care intră coeficienţii de ponderare din tabelul din bibliografie:

Weights

1	2	3	4	5	6	7	8	9	10	11	12
1	3	1	3	1	3	1	3	1	3	1	3

S = s1 + 3*s2

Cifra de control se obţine ca diferenţă între următorul număr divizibil prin 10 mai mare ca *S* şi *S*. Exemplu:

Dacă *S* = 72, următorul număr divizibul prin 10 este 80, deci cifra de control este 80-72=8.

În C++ se poate face şi altfel, având în vedere că putem calcula *restul împărţirii întregi* a lui *S* la 10:

R = S%10;

Pentru S=72 se obţine R=2. Scăzut din 10 dă tot 8.

Deci valoarea cifrei de control (a 13-a din cod) ar fi:

C = 10 - S%10.

Programul se scrie uşor (sume de valori numerice apar pe ici pe colo în curs şi aplicaţii).

```
s1 = s2 = 0;
for(i=0; i<12; i=i+2)
{
      s1 =s1 + nean13[i];
      s2 =s2 + nean13[i+1];
}
S = s1 + 3*s2;
nean13[12] = 10 - S%10;  //  Cifra de control
```

Codul final, de 13 cifre, poate fi imediat afişat pe consolă.

Desenarea codului de bare

Pentru desenarea codului de bare se va scrie o secvenţă de cod care generează un fişier de tip text care conţine codificarea imaginii codului de bare în format SVG (vezi bibliografia şi indicaţiile din enunţ).

Tabelele de codificare în binar a diferitelor cifre din codul obţinut se obţine pornind de la tabelul următor:

Structure of EAN-13

First digit	First group of 6 digits	Last group of 6 digits
0	LLLLLL	RRRRRR
1	LLGLGG	RRRRRR
2	LLGGLG	RRRRRR
3	LLGGGL	RRRRRR
4	LGLLGG	RRRRRR
5	LGGLLG	RRRRRR
6	LGGGLL	RRRRRR
7	LGLGLG	RRRRRR
8	LGLGGL	RRRRRR
9	LGGLGL	RRRRRR

Prima valoare din cod fiind intotdeauna 5, ne interesează doar linia evidenţiată. Ea arată pentru fiecare caracter din cod tabela de conversie în binar care se va utiliza (L, G pentru prima parte a codului şi R pentru partea a doua).

Encoding of the digits

Digit	L-code	G-code	R-code
0	0001101	0100111	1110010
1	0011001	0110011	1100110
2	0010011	0011011	1101100
3	0111101	0100001	1000010
4	0100011	0011101	1011100
5	0110001	0111001	1001110
6	0101111	0000101	1010000
7	0111011	0010001	1000100
8	0110111	0001001	1001000
9	0001011	0010111	1110100

Exemplu:

Poziţia	Cifra	Cod binar	Tabela folosita
[1]	5	-	-
[2]	9	0001011	L
[3]	4	0011101	G
[4]	1	0110011	G
[5]	2	0010011	L
[6]	1	etc.	L
[7]	2		G
[8]	1		R
[9]	2		R
[10]	1		R
[11]	2		R
[12]	1		R
[13]	1		R

Pentru a obţine codul binar corespunzând celor 12 valori (prima, (întotdeauna 5) nu se codifică) am putea scrie trei funcţii, fiecare trebuind să realizeze codificarea cifrelor de la 0 la 9 conform uneia dintre tabelele de codificare (L, G sau R). Codul de bare având 95 de cifre binare (precizat în temă), la fiecare apelare o astfel de funcţie ar trebui să să încarce în şirul de 95 de cifre binare un set de 7 cifre, începând de la o poziţie dată (variabila *poz*).

Exemplu de funcţie de codificare (funcţia L):

```
void L(int valoare, int bin[], int poz)
{
	switch(valoare)
	{
	case 0: //  0001101
		bin[poz]=bin[poz+1]=bin[poz+2]=bin[poz+5]=0;
		bin[poz+3]=bin[poz+4]=bin[poz+6]=1;
		break;
	case 1: //  0011001
		bin[poz]=bin[poz+1]=bin[poz+4]=bin[poz+5]=0;
		bin[poz+2]=bin[poz+3]=bin[poz+6]=1;
		break;
	case 2: //  0010011
		bin[poz]=bin[poz+1]=bin[poz+3]=bin[poz+4]=0;
		bin[poz+2]=bin[poz+5]=bin[poz+6]=1;
		break;
	case 3: //  0111101
		bin[poz]=bin[poz+5]=0;
		bin[poz+1]=bin[poz+2]=bin[poz+3]=bin[poz+4]=bin[poz+6]=1;
		break;
	case 4: //  0100011
		bin[poz]=bin[poz+2]=bin[poz+3]=bin[poz+4]=0;
```

```
            bin[poz+1]=bin[poz+5]=bin[poz+6]=1;
            break;
      case 5: //  0110001
            bin[poz]=bin[poz+3]=bin[poz+4]=bin[poz+5]=0;
            bin[poz+1]=bin[poz+2]=bin[poz+6]=1;
            break;
      case 6: //  0101111
            bin[poz]=bin[poz+2]=0;
            bin[poz+1]=bin[poz+3]=bin[poz+4]=bin[poz+5]=bin[poz+6]=1;
            break;
      case 7: //  0111011
            bin[poz]=bin[poz+4]=0;
            bin[poz+1]=bin[poz+2]=bin[poz+3]=bin[poz+5]=bin[poz+6]=1;
            break;
      case 8: //  0110111
            bin[poz]=bin[poz+3]=0;
            bin[poz+1]=bin[poz+2]=bin[poz+4]=bin[poz+5]=bin[poz+6]=1;
            break;
      case 9: //  0001011
            bin[poz]=bin[poz+1]=bin[poz+2]=bin[poz+4]=0;
            bin[poz+3]=bin[poz+5]=bin[poz+6]=1;
            break;
      }
}
```

Construirea şirului *b* de 95 de valori binare se va realiza astfel:

```
      int b[95];
      //  Se codifica inceputul: 101
      b[0]=b[2]=1;
      b[1]=0;
      //   Se codifica prima parte a codului (6 caractere, nean13[1] la
nean13[6]):
      L(nean13[1], b, 3);  //  Primul, nean13[1]. Se apeleaza functia L
      G(nean13[2], b, 10);  //  Car. 2, se apeleaza functia G
      G(nean13[3], b, 17);  //  Car. 3, se apeleaza functia G
      L(nean13[4], b, 24);  //  Car. 4, se apeleaza functia L
      L(nean13[5], b, 31);  //  Car. 5, se apeleaza functia L
      G(nean13[6], b, 38);  //  Car. 6, se apeleaza functia G
      //  Se codifica zona de separare din mijloc, 01010:
      b[45]=b[47]=b[49]=0;
      b[46]=b[48]=1;
      //  Se codifica partea a doua a codului (nean13[7] la nean13[12]).
      //  Pentru toate se apelează funcţia R():
      for(i=0; i<6; i++)
```

```
            R(nean13[7+i], b, 50+i*7);

      //  Se codifica partea finala a codului, 101:
      b[92] = b[94] = 1;
      b[93] = 0;
```

Notă: Puteţi scrie mai simplu funcţiile de codificare dacă înaintea începerii construirii şirului de valori în binar umpleţi şirul cu 0. Atunci în funcţii va trebui să scrieţi numai liniile care impun valorile de 1.

Desenarea codului de bare

Pentru desenarea codului de bare se va scrie o secvenţă de cod care generază un fişier *.html* (practic un fişier de tip text, vezi cursul 7) care conţine descrierea imaginii codului de bare folosind codificarea SVG (vezi bibliografia şi indicaţiile din enunţ).

Generarea codului SVG se va realiza pornind de la reprezentarea binară a codului. Astfel pentru fiecare cifră '1' din reprezentarea binară se va include în fişierul *.html* o comandă de trasare *<line>*. În soluţia prezentată în continuare trasarea s-a început de la poziţia x=10px (pixeli, puncte de pe ecran), y=20px iar înălţimea liniilor a fost de 30px. După fiecare poziţie binară s-a avansat cu 2px.

```
fstream cod;
cod.open( "cod.html", ios::out);
cod << "<!DOCTYPE html>" << endl;
cod << "<html>" << endl;
cod << "<body>" << endl;
cod << "<svg height=\"50\" width=\"200\">" << endl;
//  Doi pixeli pentru fiecare bara
int pozx = 10;  //  De la acest x incep trasarea
for(i=0; i<95; i++)
{
      if(b[i] == 1)   //  Se traseaza o linie
            cod << "<line x1=\"" << pozx << "\" y1=\"20\" x2=\"" << pozx
            << "\" y2=\"50\" style=\"stroke:rgb(0,0,0); stroke-width:2\" />"
<< endl;
      pozx = pozx + 2;  //  Avans cu 2px, indiferent daca s-a trasat sau nu
}
cod << "</svg>" << endl;
cod << "</body>" << endl;
cod << "</html>" << endl;
cod.close();
```

Observaţie: Pentru a introduce în şirurile de caractere scrise pe disc caracterul " (ghilimele), caracterul a fost precedat de \ (backslash) (vezi cursul 2, tipul *char*). Din acest motiv lizibilitatea codului lasă de dorit.

Exemplu:

Pentru a include linia:

<svg height="50" width="200">

s-a scris:

```
cod << "<svg height=\"50\" width=\"200\">" << endl;
```

Rezultat posibil (cod:5941212121211):

Dacă aveţi un telefon cu cameră video şi o aplicaţie de citire a codurilor de bare (de exemplu *Barcode Scanner*, disponibilă gratuit pentru Android) puteţi citi codul de bare.

Indicaţii finale:

- Analizaţi cu atenţie cerinţele folosind bibliografia indicată în enunţ;
- Rezolvaţi problema pas cu pas. Dacă nu puteţi avansa, studiaţi indicaţiile şi încercaţi să le aplicaţi. După parcurgerea unei etape încercaţie de asemenea să mutaţi codul scris într-o funcţie separată. Nu vă grăbiţi! Profitaţi cât mai mult de problema propusă pentru a înţelege modul de dezvoltare a unei aplicaţii.
- Nu încercaţi să rezolvaţi o parte care conţine noţiuni pe care nu le-aţi înţeles sau chiar nu le-aţi parcurs. Reluaţi partea din curs sau aplicaţii la care se face apel şi acum, după finalizarea studiului limbajului C++ veţi înţelege mult mai uşor.
- În programarea unei aplicaţii se foloseşte cam tot ce s-a predat. Nu trebuie să reproduceţi nimic din memorie însă trebuie să înţelegeţi fiecare noţiune, să ştiţi la ce serveşte şi unde sunt exemplele în care este folosită.

Succes!

P.S. Dacă doriţi să vedeţi şi altă abordare a limbajului C++ puteţi oricând căuta un tutorial pe ***youtube.com****. Într-o astfel de căutare am descoperit de exemplu o serie de tutoriale video scurte şi simpatice (3.5 minute fiecare!) primul fiind la adresa:*

http://www.youtube.com/watch?v=eNS0WNg1GcE