Házi Feladat

Programozás alapjai 2

Élő Dénes

LA7VEL

2015. május 10.

Tartalom:

1. Feladatkiírás:

Készítsen objektumot n-ből 1 kódú egész számok tárolására!

Valósítsa meg az összes értelmes műveletet operátor átdefiniálással (overload), de nem kell ragaszkodni az összes operátor átdefiniálásához! Amennyiben lehetséges használjon iterátort!

Legyen képes az objektum perzisztens viselkedésre!

Specifikáljon egy egyszerű tesztfeladatot, amiben fel tudja használni az elkészített adatszerkezetet! A tesztprogramot külön modulként fordított programmal oldja meg!

A megoldáshoz NE használjon STL tárolót vagy algoritmust!

A tesztprogramot úgy specifikálja, hogy az parancssoros batch alkalmazásként (is) működjön, azaz ne tételezzen fel semmilyen speciális be/kiviteli eszközt, a szabványos be/kimenetet ill. a hibakimenetet úgy kezelje, hogy az átirányítható legyen fájlba. Amennyiben a feladat lehetővé teszi, érdemes a gtest_lite ellenőrző csomagot használni a teszteléshez.

2. Pontosított specifikáció:

A házi feladatom során egy olyan adatszerkezetet kell elkészítenem, amely alkalmas 1 az n-ből kódolású egész számok tárolására. Ezen felül operátor túlterheléssel meg kell valósítanom azokat a műveleteket amik az adott objektumon elvégezhető.

Az 1 az n-ből kódolásnál a bitsorozat csak egyetlen 1-est tartalmaz. Amennyi a szám decimális értéke annyiadik bit 1-es a sorozatba.(Pl: 0=0001, 4=10000)

Mivel nem volt előre definiálva milyen műveleteket kell megvalósítani, ezért én az automatikusan létrejövő tagfüggvények mellet következőket fogom létrehozni:

- összeadás
- kivonás
- komparálás

Az osztályon belül egy sztringben fogom tárolni (ez még nem teljesen biztos) az adott bitsorozatot.

Amikor az értékadás operátorral megadjuk a kívánt sorozatot, akkor az operátor megvizsgálja, hogy az valóban a megfelelő kódolású e, tehát csak egyetlen 1-es bitet tartalmaz.

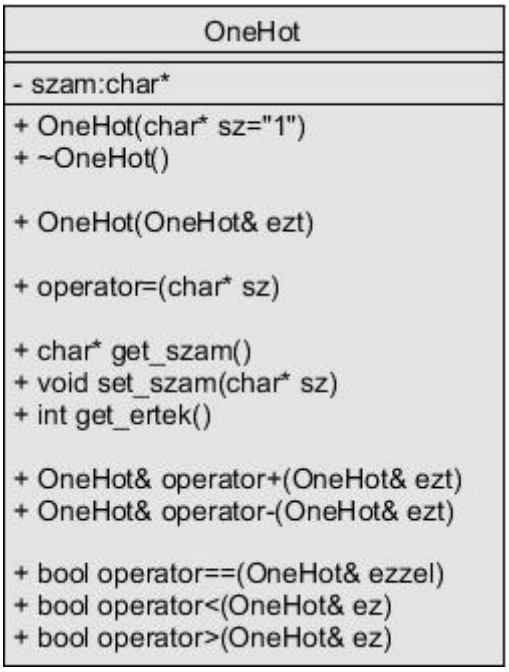
Mivel negatív számokat nem tudunk 1 az n-ből kódolással ábrázolni ezért a kivonás művelet is hibát dob ha kisebb számból vonunk ki nagyobbat.

A tesztprogramomban létre fogok hozni különböző bitsorozatokat és minden műveletet kipróbálok. Persze lesz olyan is ahol a programnak hibát kell dobnia, hiszen nem megfelelő értékeket adtam meg.

3. Terv:

3.1 Objektum terv:

A házi feladatom egyetlen adatszerkezetből fog állni. Ez az osztály tartalmaz egy privát *char* szam* adattagot, amiben az adott számot tárolja, természetes dinamikusan. A feladatom során megírom az osztály konstruktorait és destruktorát is, hogy jól használhatóak legyenek. Ahhoz, hogy 1 az n-ből kódolású számokkal el tudjuk végezni a specifikációban leírt műveleteket (összeadás, kivonás, komparálás) át kellet definiálni a logikus operátorokat. Az osztály több tagfüggvénnyel is rendelkezik, amelyekkel leginkább az adattagot lehet lekérdezni vagy módosítani.



3.2 Algoritmusok:

A konstruktornál és a *set:szam(char* sz)* függvénynél meg kell vizsgálni, hogy a paraméterben kapott sztring pontosan egy darab egyest tartalmaz. Ha nem akkor olyan kivételt kell dobni, hogy *nem 1 az n-ből kódolású a szám*.

A –operátornál azt kell megvizsgálni, hogy mindig a nagyobb számból vonjuk ki a kisebbet mert 1 az n-ből kódolással nem tudunk negatív számokat ábrázolni. Ilyenkor dobunk egy kivételt, hogy *érvénytelen művelet*.

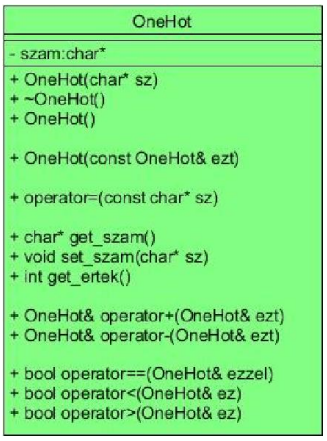
3.3 Tesztelés:

A tesztelés során lesznek példányok, amelyeknek lesz kezdőértéke, de lesz amelyeknek a standard inputról fog érkezni az értéke. Létrehozok néhány példányt az adott osztályból és műveletek sorozatával tesztelem az adatszerkezet összes tagfüggvényét és konstruktorát.

4. Végleges program:

A végleges házi feladatom egy *calss.h*-ból, egy *class.cpp*-ből és egy *OneHot.cpp*-ből áll.

A *class.h* az adott osztály header fájlja. Ebben a fájlban deklarálva van a OneHot osztály összes tagfüggvénye, konstruktorai illetve a destruktor.



(végleges osztálydeklaráció)

Az osztálydefiníciót a class.cpp fájl tartalmazza. Ebben a konstruktorok és tagfüggvények mellet még két függvény szerepel. A *stringcpy()* és az *ellenoriz()*.

A stringcpy függvény ugyan úgy működik mit a beépített strcpy függvény, csak azért kellet megírni mert ismeretlen okok miatt nem volt hajlandó lefutni a beépített függvény. A függvény első paramétere a cél sztring, a második paramétere a forrás sztring.

Az ellenoriz függvénynek csak egy paramétere van, az a szting amit ellenőrizni szeretnénk. A függvény megvizsgálja, hogy a kapott adat megfelel e az 1 az n-ből kódolásnak. Tehát csak 1-est és 0-át tartalmazz és 1-esből is csak egy darab van a bitsorozatban. Ha a feltételek teljesülnek akkor igazzal tér vissza ellenkező esetben hamissal.

Az osztály rendelkezik egy egyparaméteres, és egy paraméter nélkülikonstruktorral. A paraméter nélküli egy üres sztringet hoz létre. Az egyparaméteres a paraméterül kapott bitsorozatot először átadja az ellenoriz függvénynek. Ha igaz értékkel tér vissza a másoló függvénnyel beállítja a bitsorozatot. Ha hamis értékkel tér vissza akkor dob egy szöveges hibát miszerint hibás kódolású.

Az össze tagfüggvény amelyik vár egy sztringet paraméterül, annál lefut az ellenörző függvény és ha nem helyes akkor ugyanúgy azt a hibát dobja, hogy „Incorrect coding”.

A már létrehozott OneHot objektumnak új értéket tudunk adni a *set_szam()* függvénnyel vagy az = operátorral. Mind a kettő egy const char*-ot vár paraméterül és ezt állítja be új értéknek.

A program során le tudjuk kérdezni az objektum értéket mint 1 az n-ből kódolás, vagy mint decimális szám. Előbbit a *get_szam()* utóbbit a *get_ertek()* nevű függvénnyel tehetjük meg.

Az objektumhoz tartoznak relációs operátorok: <, >, ==. Mindegyik egy bool típussal tér vissza. Mind a három operátor meghívja a get_ertek nevű függvényt és az általa visszaadott értékeket hasonlítja össze.

5. Tesztelés:

A teszteseteket a OneHot.cpp fájl tartalmazza. Összesen 6 teszt fut le, amelyek során az összes lehetséges függvény és operátor kipróbálásra kerül.

Teszt 1:

```
void teszt1()
{
 cout<<"##########TESZ 1##########"<<endl<<endl;

    try{

    OneHot t1("010000000000");
    OneHot t2("0100");

    OneHot t3=t1-t2;

    cout<<t1.get_szam()<<" "<<t2.get_szam()<<" "<<t3.get_szam()<<endl;
    }


    catch(const char* error) {cout<<error;}
    cout<<endl<<endl;
}
```

Az első teszt során létrehozok két objektumot, az egyparaméteres konstruktorral. Ezt követően létrehozok egy harmadik objektumot is aminek az = oprátorral adok értéket mégpedig az előző kettő különbségét. Végül már csak mindegyiket kiírom 1 az n-ből kódolással.

Teszt 2:

```
void teszt2()
{
    cout<<"##########TESZ 2##########"<<endl<<endl;

    try
    {

    OneHot t1("010000"); //4
    OneHot t2("0010000000000000000");//15

    OneHot t3=t1+t2;;

    cout<<t1.get_ertek()<<" "<<t2.get_ertek()<<" "<<t3.get_ertek()<<endl;

    }
    catch(const char* error)    {cout<<error;}

    cout<<endl<<endl;

}
```

Ebben a tesztben ugyan azt csinálom mint az elsőben csak a kivonás helyett itt összeadás szerepl.

Teszt 3:

```
void teszt3()
{
    cout<<"##########TESZ 3##########"<<endl<<endl;

    try
    {
        OneHot t1;

        t1.set_szam("01000");//3
        cout<<t1.get_szam()<<"||"<<t1.get_ertek()<<endl;
        t1.set_szam("0110000");//HIBA
        cout<<t1.get_szam()<<"||"<<t1.get_ertek()<<endl;

    }
    catch(const char* error)    {cout<<error;}

    cout<<endl<<endl;
```

A hármas tesztben vizsgálom a hibakezelést is. Először létrehozok egy t1 objektumot jó értékkel. Ezt kiírom mint decimális mint 1 az n-ből kódolású formában is. Ezt követően adok neki egy új értéket ami nem csak egy darab 1-est tartalmaz így a tesz során hiba keletkezik amire az *„Incorrect coding ”* hívja fel a figyelmünket.

Teszt 4:

```
void teszt4()
{
    cout<<"##########TESZ 4##########"<<endl<<endl;

    try
    {

    OneHot t1;
    cout<<t1.get_ertek()<<endl;
    t1="1000";//3
    cout<<t1.get_ertek()<<endl;
    OneHot t2(t1);
    cout<<t2.get_szam()<<"||"<<t2.get_ertek()<<endl;

    }
    catch(const char* error)    {cout<<error;}

    cout<<endl<<endl;

}
```

A 4-es teszt során használom a másolókonstruktort és a paraméternélküli konstruktort is. Létrehozok egy t1 objektumot, aminek meghívódik a paraméternélküli konstruktora. Ilyenkor egy ’\0’-t tartalmazó sztring jön létre. Ezután kiírom a t1 decimális értékét ami egy invalid szám hiszen még nem adtunk meg neki pontos értéket. Ezt az = operátorral korrigáljuk, és ezt követően az újonnan létrehozott t2 objektumba másoljuk a t1-et.

Teszt 5:

```
void teszt5()
{
cout<<"##########TESZ 5##########"<<endl<<endl;

    try
    {

    OneHot t1("01000");//3
    OneHot t2("10000000");//7

    cout<<boolalpha<<(t1<t2)<<endl; //3<7
    cout<<boolalpha<<(t1>t2)<<endl; //3>7
    cout<<boolalpha<<(t1==t2)<<endl;//3==7


    }
    catch(const char* error)    {cout<<error;}

    cout<<endl<<endl;

}
```

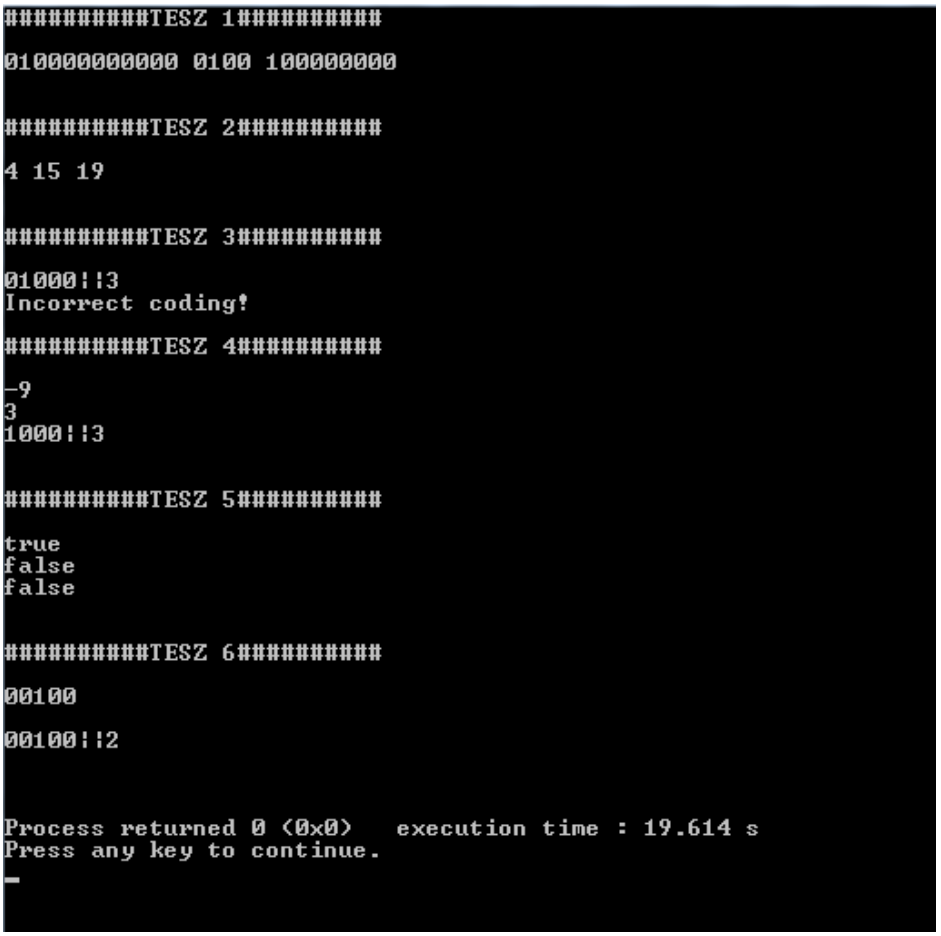
Az ötödik teszt során használom a komparátorokat. Létrehozok két különböző objektumot, mind a kettőt az egyparaméteres konstruktorral. Ezt követően kiírom a kimenetre a három komaparálás által visszaadott bool értéket.

Teszt 6:

```
void teszt6()
{
cout<<"##########TESZ 6##########"<<endl<<endl;

    try
    {
        char pl[256];

       cin.getline(pl,256);

        OneHot t1;
        t1=pl;

      cout<<endl<<t1.get_szam()<<"||"<<t1.get_ertek()<<endl;

    }
    catch(const char* error)    {cout<<error;}

    cout<<endl<<endl;
}
```

Az utolsó teszt során nem én adok meg egy kezdőértéket, hanem a bementről kell megadni egy maximum 256 hosszú bitsorozatot. Ezt követően ezzel a bitsorozattal létrehozok egy t1 objektumot, majd kiírom a decimális és az 1 az n-ből kódolású értékét is.

A hat teszteset lefuttatása után, ha a hatodik tesztben bemenetnek a „00100” bitsorozatot adom meg , akkor a következő kimenetet kapjuk:



Mindegyik teszt lefutása után a várt kimenetet kapjuk.