Leírás

Valósítsunk meg egy egyszerű vektor típust, mely adott típusú értékek egy sorozatát tárolja! A sorozat hossza korlátos: ezt a korlátot nevezzük a vektor kapacitásának. Ez azt jelenti, hogy a vektorban tárolt értékek száma (a sorozat hossza, a vektor "mérete") mindig nulla és a kapacitás közt van.

Alapfeladat (13 pont)

Készítsük el a `vector` típust egy rekorddal (struktúrával) ábrázolva. A `vector` legyen egy típusszinoníma a struktúra típusra. Tartsuk nyilván a vektor kapacitását (nemnegatív egész), pillanatnyi méretét (nemnegatív egész) és egy mutatót egy egybefüggő memóriaterületre, mely kapacitás számú elemet képes tárolni. A vektor elemeinek a típusa legyen `int`, de a kód olvashatóságának és módosíthatóságának növelése érdekében vezessünk be egy makrót az elemtípusra. Definiáljunk négy műveletet a `vector` típushoz. - `initialize(v,capacity)` : inicializálja a vektort a megadott kapacitásnak megfelelően. A vektor kezdetben üres, egy elemet sem tartalmaz. A függvény visszaadja, hogy sikerült-e az inicializálás. Az inicializálás sikertelen legyen, ha a megadott kapacitás nem pozitív. - `dispose(v)` : felszabadítja a vektor által birtokolt dinamikus memóriát, és a vektor kapacitását nullára állítja. - `append(v,e)` : ha még nem telt meg a vektor (a méret kisebb, mint a kapacitás), egy elemet betesz a sorozat végére. A függvény visszaadja, hogy sikerült-e az elemet betenni a vektorba. - `retrieve(v,i)` : a függvény visszaadja a sorozat adott indexű elemét. Feltehetjük, hogy a megadott index (amely egy nemnegatív egész szám) érvényes, azaz kisebb, mint a vektor mérete (a tárolt sorozat pillanatnyi hossza). Készítsünk főprogramot a `vector` típus letesztelésére! Elegendő "beégetett" tesztet írni, nem kell fájlból vagy szabványos bemenetről olvasni, vagy a parancssori argumentumokat használni. Figyeljünk arra, hogy a programban ne legyen memóriaszivárgás!

Menő főprogram (4 pont)

Készítsünk olyan főprogramot, amely a parancssori argumentumokkal vezérelten képes letesztelni a `vector` típust! Feltehető, hogy a parancssori argumentumok értelmesek, azaz nem kell az input ellenőrzésével foglalkozni. Ha a programot így futtatjuk:

```
./test_vector 5 a 3 r 0 append 9 retrieve 1 aaaaa 8 r 0 r 2
```

akkor ez azt jelenti, hogy inicializálunk egy vektort 5 kapacitással, betesszük a 3 értéket, lekérdezzük a 0 indexű értéket, betesszük a 9 értéket, lekérdezzük az 1 indexű értéket, betesszük a 8 értéket, végül lekérdezzük a 0 indexű és a 2 indexű értéket. Amint látjuk, minden második parancssori argumentumot az első betűje alapján parancsnak fogjuk értelmezni, az utána következő parancssori argumentumot pedig a parancs paraméterének.

A főprogram minden parancs elvégzése után írja ki a parancs (szignifikáns) első betűjét és a parancs eredményét (a visszaadott értéket) egy új sorba. (És természetesen más függvényből nem írogatunk ki semmit a kimenetre...)

Modularizáció (4 pont)

Bontsuk fel a programkódot több fordítási egységre a tanultaknak megfelelően! A modulok közötti interfész leírására használjuk fejállományt (header-fájlt), amelyben a konvenciókat követő include-guard található.

Beszúrás és törlés (4 pont)

Valósítsunk meg további két műveletet a vector típushoz! - `insert(v,i,e)` : beszúr egy elemet a vektorba a megadott pozícióra, és visszaadja, hogy sikerült-e. A beszúrás akkor lehetséges, ha a vektor nincs még megtelve, és a megadott index értelmes. Például egy 10 kapacitású, 5 hosszú vektorba a 7. indexpozícióra azért nem lehet elemet beszúrni, mert az 5. és 6. indexpozíción "lyukas" lenne a sorozat. A beszúrás során a megadott indexpozícióval kezdődően az elemeket eggyel hátrébb kell másolni! - `erase(v,i)` : törli a megadott indexű elemet a sorozatból, és visszaadja, hogy sikerült-e. A törlés akkor lehetséges, ha az indexpozíción tényleg található elem a sorozatban (azaz a megadott index kisebb, mint a méret). A törlés során keletkezett lyukat a későbbi elemek előrébb mozgatásával el kell tüntetni. Módosítsuk a főprogramot is!

Kapacitásbővítés (5 pont)

Valósítsunk meg további három műveletet a `vector` típushoz! - `set_capacity(v,c)` : megváltoztatja a vektor kapacitását a megadott értékre, és visszaadja, hogy ez sikerült-e. Ehhez az kell, hogy az új kapacitásérték legalább akkora legyen, mint a sorozat mérete. Foglaljunk le egy új egybefüggő memóriaterületet az új kapacitásértéknek megfelelően, és mozgassuk át a régi területről az adatokat az újra. A mozgatáshoz készítsünk egy segédfüggvényt, amely csak a tartalmazó fordítási egységben hívható meg! - `safe_append(v,e)` : mint az `append(v,e)`, de ha a vektor már meg van telve, próbálja meg a `set_capacity` segítségével duplájára növelni a kapacitást. - `safe_insert(v,i,e)` : mint az `insert(v,i,e)`, de ha a vektor már meg van telve, próbálja meg a `set_capacity` segítségével duplájára növelni a kapacitást.