4. Verem adatszerkezet - Stack

4.1 Verem definiálása

Definíció 4.1.1 A verem egy olyan adatszerkezet, amely sorrendben tartja az elemeket, és a legutolsóként hozzáadott elem mindig az első, amely eltávolításra kerül (ezt gyakran **LIFO - Last In First Out** - rendszerként is emlegetik).

A veremhez hozzáadni (push) és eltávolítani (pop) elemeket lehet, de közvetlen hozzáférés nincs az elemek másik részéhez. Az üres verem állapota jellemzően az, amikor egyetlen elem sem található benne, de az is előfordulhat, hogy a verem telített (amikor az adatszerkezet kapacitása megtelt). A verem gyakran alkalmazott adatszerkezet például a programozásban, az algoritmusokban vagy a memóriakezelésben. A verem típusának implementálásához C programozási nyelvben általában tömbök használatával valósítják meg.

4.2 Alapvető műveletek

1. **Push**: Ezzel a művelettel új elemet helyezünk a verem tetejére (azaz legutolsó elemként) a meglévő elemek fölé.
2. **Pop**: Ezzel a művelettel eltávolítjuk a verem tetején lévő elemet. Az eltávolított elem visszatérhet a hívó programhoz, és a verem teteje eggyel lejjebb kerül.
3. **Top**: Ezzel a művelettel lekérdezhetjük a verem tetején lévő elemet, anélkül, hogy eltávolítanánk azt a veremből.
4. **Empty**: Ezzel a művelettel ellenőrizhetjük, hogy a verem üres-e vagy sem.
5. **Full**: Ez a művelet nem egy külön álló művelet, hanem inkább az adatszerkezet kapacitásának a telítettségét jelzi. Ha a verem már elért egy adott kapacitást, akkor azt mondjuk, hogy a verem "telített" vagy "full". Azaz, amikor a verembe már nem lehet újabb elemet hozzáadni a kapacitás korlátai miatt. A verem kapacitása általában fix, vagyis a verem elemeinek száma nem növekszik és nem csökken dinamikusan. Ha a verem telített, akkor új elemeket csak akkor lehet hozzáadni, ha a verem tetején lévő elemek eltávolításra kerülnek.
6. **Size**: Ez a művelet a veremben tárolt elemek számát adja vissza és arra szolgál, hogy lekérdez-

zük a verem aktuális méretét. Ha a verem üres, akkor a "size" művelet -1 értékkel tér vissza. Ha a veremben van legalább egy elem, akkor a "size" művelet visszatér az aktuális elemek számával. A veremben tárolt elemek száma általában korlátozott, hiszen a verem kapacitása is meghatározott. Azaz, ha a verem telített, akkor a "size" művelet visszatér a verem maximális méretével.

Ezen kívül, a verem adatszerkezet egyéb műveletei is lehetnek, például a méret lekérdezése (size), a verem másolása (copy), vagy az adott elem megtalálása a veremben (search). Azonban a push, pop, top és empty a verem alapvető műveletei.

4.3 Tulajdonságok

1. **LIFO**: A veremben tárolt elemek sorrendje LIFO (Last In First Out), ami azt jelenti, hogy az utolsóként hozzáadott elem az első, amely eltávolításra kerül.
2. **Korlátozott hozzáférés**: A veremben csak a tetején található elem érhető el közvetlenül, más elemekhez csak a push és pop műveletek segítségével férhetünk hozzá.
3. **Gyors beszúrás és eltávolítás**: A veremben az elemek hozzáadása (push) és eltávolítása (pop) konstans időben történik, így a verem használata hatékony.
4. **Nem támogatja a közvetlen hozzáférést**: A veremben nem lehet közvetlenül hozzáférni más elemekhez, csak a tetején található elemhez.
5. **Nem dinamikus**: A verem általában fix méretű, ami azt jelenti, hogy az elemek száma nem változik dinamikusan.
6. **Alkalmas verem alapú algoritmusok megvalósítására** : A verem adatszerkezet különösen alkalmas olyan algoritmusok megvalósítására, amelyekben fontos a LIFO sorrend, például a szintaxisfák, az alárendelt függvények hívása vagy a visszalépési pontok nyilvántartása során.

4.4 Veremmel kapcsolatos függvények

Definiálunk egy Stack struktúrát, amely egy verem (stack) adatszerkezetet reprezentál. A struktúra három adattagot tartalmaz:

- **capacity**: Az adatszerkezetben tárolható maximális elemkapacitást jelöli. Ez az érték előre megadott MAX SIZE értékből származik, amely az adatszerkezet méretét határozza meg

- **top**: Az adatszerkezetben tárolt elemek legfelső (csúcs) pozícióját jelöli. Ha az adatszerkezet üres, akkor az értéke -1 lesz. Amikor az adatszerkezetben egy új elem kerül eltárolásra, akkor az adatszerkezet csúcsa megnövekszik 1-gyel

- **elements**: Az adatszerkezetben tárolt elemek tömbje. A tömb elemszáma a capacity adattag által meghatározott értékkel megegyező, és az adatszerkezet által tárolt összes elemet tartalmazza. Az elemek hozzáadása és eltávolítása a verem műveletek (push(), pop()) segítségével történik. Az elements tömb az int típusú elemeket tárolja

```
typedef struct {
    int capacity;
    int top;
    int *elements;
}Stack;
```

4.4.1 Helyfoglalás

```
void createStack(int capacity, Stack *stack) {
    stack->capacity = capacity;
    stack->top = -1;
    stack->elements = (int*)calloc(stack->capacity, sizeof (int));
    if(!stack->elements) {
        printf(MEMORY_ALLOCATION_ERROR_MESSAGE);
        exit(MEMORY_ALLOCATION_ERROR_CODE);
    }
}
```

A függvény kap egy kapacitás értéket és egy üres verem (stack) mutatót. Az inicializálás során a függvény beállítja a verem kapacitását, a verem csúcsának (top) kezdeti értékét -1-re (mivel az üres veremnél a csúcs mutatója érvénytelen), és lefoglalja a verem elemeinek tárolására szolgáló memóriát a kapacitásnak megfelelően. Ha a memóriafoglalás sikertelen, akkor a függvény kiír egy hibaüzenetet és kilép a programból.

4.4.2 Üres verem tesztelése

```
bool isEmpty(Stack stack) {
    return stack.top == -1;
}
```

A függvény egy verem adatszerkezetet vár bemenetként, majd visszatérési értékként igazat (true) ad vissza, ha a verem csúcsa (top) értéke -1, azaz nincsenek elemei a veremben. Ellenkező esetben hamisat (false) ad vissza, mert a veremben vannak elemek.

A függvény hasznos lehet a verem műveletek, mint például a pop() és a peek() műveletek előtt, mivel ellenőrzés nélkül azok nem biztonságosak az üres veremekre. Az isEmpty() függvény segítségével ellenőrizni tudjuk, hogy van-e még elem a veremben, mielőtt azokat eltávolítanánk vagy lekérdeznénk a verem tetejéről.

4.4.3 Telített verem tesztelése

```
bool isFull(Stack stack) {
    return stack.top == stack.capacity-1;
}
```

A függvény egy verem adatszerkezetet vár bemenetként, majd visszatérési értékként igazat (true) ad vissza, ha a verem csúcsa (top) értéke megegyezik a verem kapacitásával (capacity) csökkentett egyel. Ez azt jelenti, hogy az adatszerkezetben már nincs több hely az új elemek számára, és a további push() műveletek nem fognak működni. Ellenkező esetben hamisat (false) ad vissza, mert az adatszerkezetben még van hely az új elemek számára.

A függvény hasznos lehet a verem műveletek, mint például a push() művelet előtt, mert így ellenőrizhető, hogy az adatszerkezetben van-e még hely az új elemek számára. Ha az adatszerkezet megtelt, akkor az új elemeket már nem lehet hozzáadni a veremhez, és a push() művelet nem lesz sikeres.

4.4.4 Elem beszúrása - Push

```
void push(Stack *stack, int item) {
    if(isFull(*stack)) {
        printf(FULL_MESSAGE);
        return;
    }
    stack->elements[++stack->top] = item;
}
```

A függvény két bemeneti paramétert vár: a verem mutatóját (stack) és az új elemet (item). A push() függvényben először a isFull() függvényt használjuk az adatszerkezet megtelt állapotának ellenőrzésére. Ha az adatszerkezet megtelt, akkor a függvény kiírja a megfelelő üzenetet a felhasználónak, majd visszatér a függvényből, anélkül hogy hozzáadná az új elemet a veremhez.

Ha az adatszerkezet nem telt meg, akkor a függvény hozzáadja az új elemet a veremhez a következő lépésekkel:

Az adatszerkezet csúcsát (top) növeli 1-gyel (++stack->top), hogy a csúcs a következő üres helyre mutasson a tömbben. Az új elemet (item) eltárolja az adatszerkezetben a megfelelő pozícióban (stack->elements[stack->top] = item). Végül, ha a push() függvény sikeresen hozzáadta az új elemet az adatszerkezethez, akkor a verem tartalmazni fogja az új elemet a tetején.

4.4.5 Elem eltávolítása - Pop

```
int pop(Stack *stack) {
    if (isEmpty(*stack)) {
        printf(EMPTY_MESSAGE);
        return INT_MIN;
    }
    int save = stack->elements[stack->top];
    stack->elements[stack->top--] = 0;
    return save;
}
```

A függvény egy bemeneti paramétert vár: a verem mutatóját (stack). Először az isEmpty() függvényt használjuk az adatszerkezet ürességének ellenőrzésére. Ha az adatszerkezet üres, akkor a függvény kiírja a megfelelő üzenetet, majd visszatér a függvényből a minimum int értékkel, hogy jelezze, hogy nem volt értékes elem az adatszerkezetben.

Ha az adatszerkezet nem üres, akkor a függvény eltávolítja a tetején lévő elemet a következő lépésekkel:

Az adatszerkezet tetején lévő elemet (stack->elements[stack->top]) eltároljuk egy ideiglenes változóban (save). Az eltávolított elem helyét 0-ra állítjuk (stack->elements[stack->top–] = 0). Az adatszerkezet csúcsát (top) csökkentjük 1-gyel, hogy a csúcs a következő elemre mutasson. Végül, ha a pop() függvény sikeresen eltávolította az elemet az adatszerkezetből, akkor a függvény visszatér az eltávolított elemmel (save), ami a tetején lévő elem volt.

4.4.6 Legfelső elem visszetérítése - Peek vagy Top

```
int peek(Stack stack) {
    if (isEmpty(stack)) {
        printf(EMPTY_MESSAGE);
```

```
        return INT_MIN;
    }
    return stack.elements[stack.top];
}
```

A függvény egy bemeneti paramétert vár: a verem adatszerkezetet (stack). Először a isEmpty() függvényt használjuk az adatszerkezet ürességének ellenőrzésére. Ha az adatszerkezet üres, akkor a függvény kiírja a megfelelő üzenetet, majd visszatér a függvényből a minimum int értékkel, hogy jelezze, hogy nem volt értékes elem az adatszerkezetben.

Ha az adatszerkezet nem üres, akkor a függvény visszaadja az adatszerkezet tetején lévő elemet a következő lépésekkel:

Az adatszerkezet tetején lévő elemet (stack.elements[stack.top]) adja vissza. Végül, ha a peek() függvény sikeresen visszaadta az elemet az adatszerkezetből, akkor a függvény visszatér az elemmel, ami a tetején lévő elem volt, anélkül, hogy eltávolította volna azt.

4.4.7 Elemek száma - Size

```
int size(Stack stack) {
    return stack.top+1;
}
```

A függvény egy bemeneti paramétert vár: a verem adatszerkezetet (stack). A méret kiszámítása egyszerű: csak ki kell számolni az adatszerkezetben lévő elemek számát, ami egyenlő az adatszerkezet tetején lévő elem indexével (stack.top) plusz egy. Ennek az az oka, hogy az indexek nullától kezdődnek az adatszerkezetben, így ha az utolsó elem indexe például 4, akkor összesen 5 elem van az adatszerkezetben.

Végül a függvény visszatér az adatszerkezet méretével.

4.5 Megoldott feladatok

Adott egy verem, amelybe legtöbb 20 egész számot tehetünk és amelyhez csak akkor lehet hozzáadni egy x számot, ha a verem tetején található szám nagyobb, mint x. Ha ez a feltétel nem teljesül, akkor a verem elemei eltávolításra kerülnek, amíg az adott feltétel teljesül, vagy amíg a verem üres nem lesz. Például, ha hozzá szeretnénk adni az 20,5,16,9,3,7,5,4,8 számokat egy kezdetben üres veremhez, akkor mi történik? Hány elem lesz a végén a veremben?

Megoldás:

```
Stack stack;
    createStack(20,&stack);
    if(!freopen("szamsor.txt","r",stdin))
    {
        printf("File not found");
        exit (-2);
    };
    int x;
    while(scanf("%i",&x) != EOF)
    {
        if(isEmpty(stack)){
            push(x, &stack);
```

```
        }
        else
        {
            while(peek(stack)<x && !isEmpty(stack))
            {
                pop(&stack);
            }
            push(x,&stack);
        }
    }
    printf("%i", size(stack));
}
```