1. Kis zárthelyi, megoldások és pontozás

a kijavított kis zárthelyik megtekinthetők a március 13. gyakorlat után

---

1. IGAZ/HAMIS

“Egy pointerrel bármennyiszer új objektumra mutathatunk, míg a referencia inicializáció után nem változtatható meg” - IGAZ (1p)

---

2. HELYES-E? (“A válasz csak indoklással fogadható el”)

a)

```
const int array []={1,2,3};
int* ptr= array;
```

NEM. Konstans elemű tömbre csak konstansra mutató pointerrel mutathatunk rá. (2p)

Ha a C++ engedné, a ptr-el meg tudnánk változtatni a konstansnak feltételezett (és emiatt pl. ROM-ban tárolt) tömb elemeit.
A helyes ez lenne: const int *ptr= array;

b)

```
int &a;
fv(a);
...
void fv(int &a) {
        a++;
 }
```

NEM. A referenciát inicializálni kell.

---

3. “Írjuk meg a stack_pop függvényt, ami egy verem (stack) adatstruktúrából visszaadja a legutoljára betett elemet, a stack int típusú elemeket tárol. Használd az alábbi adatstruktúrát (5 pont):

```
struct stack
{
       int elements;
       int *pData;
};
```

1 pont – helyes fejléc (referencia vagy pointer)
2 pont – helyes memóriakezelés (foglalás + felszabadítás)

2 pont – stack pop helyes implementálása”

Mintamegoldás:

```
int stack_pop(struct stack& s)
{
	// ures a stack?
	if (s.elements == 0) {
		// itt valamit csinalni kell. -1 -et visszaadni, mint a laboron a karakteres
		// stack eseten nem lesz jo
		// kiirunk valami hibat es kiszallunk.
		fprintf(stderr, "Stack error at stack_pop\n");
		abort();
		// majd dobunk exception-t, ha megtanultuk
	}

	int value = s.pData[s.elements - 1];// ez az utolso eleme a tombnek
	s.elements--; // eggyel csokkentjuk az elemek szamat

	if (s.elements > 0) { // meg van elem a veremben
		// uj tombot allokalunk
		int *tmp= (int*)malloc(s.elements*sizeof(int));
		// itt masolunk, ahogy a laboron volt
		for (int i = 0; i<s.elements; i++)
			tmp[i] = s.pData[i];
		// felszabaditjuk a regit
		free(s.pData);
		// az ujat hozzakotjuk
		s.pData = tmp;
	} else { // kivettuk az utolso elemet is
		free(s.pData);
		s.pData = 0;
	}
	return value;
}
```

Megjegyzések:
Teljes implementációs szabadság volt, amit a feladat nem specifikált, ha jól van megvalósítva, jár a pont.

- a laboron karakteres stack volt. Aki ilyet csinált, most -1 pont, legközelebb nem fogadom el. Egészekről szólt a feladat.
- definíciónál vagy referenciát, vagy pointert kell átadni, mivel a struktúra állapota meg fog változni. Tehát egy int stack_pop(stack s) definíció elvi hibás, hiszen ezen bármit módosítunk, visszatéréskor elveszik. Az a kisebbik gond, hogy túlcsordulhat a verem.
- elegáns definíció a bool stack_pop(stack &s, int &val)
    - így megkerüljük az üres stack esetén a mit adunk vissza kérdést
    - a laboron karakterekből állt a stack és egész visszatérési értékű volt, ott a -1 tökéletes hibajelzés. (hasonlóan pl. az int getc() -hez). Itt ez a megoldás nem lesz túl jó, mert a -1 lehet valós adat is.
- helyes memóriakezelés: lefoglal eggyel kevesebbet, másol, felszabadít, ebben a sorrendben.

    - malloc/free vagy new/delete mindegy. delete [ ] operátor kell a tömb felszabadításához
    - C++ -ban kell a cast malloc esetén
    - inteket tárolunk, darabszám*int mérete byte-ot kell foglalni malloc használatakor
- pop helyes implementálása:
    - a tömb utolsó eleme az a méret -1 . elem. Tehát ha stack *s, akkor s->pData[s->elements-1]
    - két speciális eset van, ezeket illett volna kezelni
        - a stack üres
        - a stackben csak egy érték van, ilyenkor csak felszabadítás kell. (***)

(***) Mélyvíz.
Ha a könyvtár szabványos, akkor nem kell külön speciális esetként kezelni. Nézzünk utána, mi történik, ha malloc-tól 0 byte-ot kérünk.