1. Elméleti lap

Áttekintő:

Egyszerű függvények implementálása

```
1. int sum(int number1, number2);
2. float minimum(float number1, float number2, float number3);
3. int* allocateMemoryForArray1(int n);
4. void allocateMemoryForArray2(int n, int **dpArray);
5. void readArray(int *pn, int **dpArray, const char *input);
6. void printArray(int n, int *pArray, const char *output);
7. void deallocateMemoryForArray(int **dpArray);
```

Megoldás:

Deklaráció:

```
int sum(int number1, int number2);
```

Függvény dokumentáció:

```
/**
 * Kiszámolja és visszatéríti két egész szám összegét.
 * @param number1 első szám
 * @param number2 második szám
 * @return két szám összege
 */
```

Definíció:

```
int sum(int number1, int number2)
{
    return number1 + number2;
}
```

Kódmagyarázat (leírás saját szavaiddal)

A függvény nem használ más változót, csupán egyetlen sorában visszatéríti a 2 szám összegét.

Megjegyzések (Big O):

- időbeli komplexitás **O(1)**

- térbeli komplexitás **O(1)**

Deklaráció:

```
float minimum(float number1, float number2, float number3);
```

Függvény dokumentáció:

```
**
 * Kiszámolja és visszatéríti 3 valós szám közül a legkisebbet.
 * @param number1 első szám
 * @param number2 második szám
 * @param number3 harmadik szám
 * @return a három szám közül a legkisebb (min)
 */
```

Definíció:

```
float minimum(float number1, float number2, float number3) {
    float min = number1;
    if(number2 < min)
        min = number2;
    if(number3 < min)
        min = number3;
    return min;
}
```

Kódmagyarázat

A függvény először összehasonlítja az első két paramétert, és a kisebbet eltárolja a min változóban. Ezután a min értékét összeveti a harmadik paraméterrel, így végül a három szám közül a legkisebbet adja eredményül.

Megjegyzések:

- időbeli komplexitás **O(1)**
- térbeli komplexitás **O(1)**

Deklaráció:

```
int* allocateMemoryForArray1(int n);
```

Függvény dokumentáció:

```
**
 * Helyet foglal dinamikusan egy n elemű tömbnek és visszatér a
lefoglalt memória címével
```

```
 * @param n tömb elemeinek a száma
 * @return a lefoglalt tömb kezdőcímét
 */
```

Definíció:

```
int *allocateMemoryForArray1(int n) {
    int *array=(int*)malloc(n*sizeof (int));
    if(!array)
    {
        printf("Memory allocation error");
        return NULL;
    }
    return array;
}
```

Kódmagyarázat

A függvény egy n elemű egész számokat tartalmazó tömb számára foglal helyet dinamikusan a malloc segítségével.

A lefoglalt memória címét egy int* típusú mutató (array) tárolja.

Ha a malloc meghívása sikertelen, kiírunk egy hibaüzenetet és NULL-t adunk vissza.

Ha a foglalás sikeres, a függvény visszatér a lefoglalt memória kezdőcímével, amelyet a hívó függvény felhasználhat.

Megjegyzések:

- időbeli komplexitás **O(n)**
- térbeli komplexitás **O(n)**

Deklaráció:

```
void allocateMemoryForArray2(int n, int **dpArray);
```

Függvény dokumentáció:

```
**
 * Dinamikusan helyet foglal egy n elemű egész számokat tároló tömbnek, és a lefoglalt memória címét a dpArray paraméteren keresztül adja vissza.
 * @param n tömb elemeinek a száma
 * @param dpArray dupla mutató, amely a lefoglalt tömb első elemére mutató mutatót tárolja
 */
```

Definíció:

```
void allocateMemoryForArray2(int n, int** dpArray)
{
    *dpArray = (int*)malloc(n * sizeof(int));
    if (!(*dpArray))
    {
        printf("Memory allocation error");
        exit(-1);
    }
}
```

Kódmagyarázat

A függvény egy n elemű egész számokat tartalmazó tömb számára foglal helyet dinamikusan.
A dpArray paraméter egy **mutató egy mutatóra** (int**), amely lehetővé teszi, hogy a lefoglalt memória címét a hívó függvényben is elérhetővé tegyük.
Ha a malloc nem tud memóriát foglalni, egy hibaüzenetet írunk ki és kilépünk a programból.
Ha a foglalás sikeres, a dpArray által mutatott mutató értékét beállítjuk a lefoglalt memória címére, így az a hívó függvényben is elérhető lesz.

Megjegyzések:

- időbeli komplexitás **O(n)**
- térbeli komplexitás **O(n)**

Deklaráció:

```
void readArray(int *pn, int **dpArray, const char *input);
```

Függvény dokumentáció:

```
/**
 * Beolvassa egy tömb méretét, dinamikusan helyet foglal a tömb számára, majd az elemeit is beolvassa egy bemeneti forrásból.
 * @param @pn pointer, a tömb elemeinek a száma felé mutató
 * @param @dpArray dupla mutató, amely a lefoglalt tömb első elemére mutat
 * @param @input karakterlánc, amely a bemeneti forrást jelöli
 */
```

Definíció:

```
void readArray(int* pn, int** dpArray, const char* input)
{
    if (!freopen(input, "r",stdin))
    {
        exit(-2);
    }
    scanf("%i", pn);
    allocateMemoryForArray2(*pn, dpArray);
    for (int i = 0; i < *pn; i++)
    {
        scanf("%i", &((*dpArray)[i]));
        //scanf("%i",*dpArray+i);
    }
    freopen("CON", "r",stdin);
}
```

Kódmagyarázat

A függvény beolvassa egy tömb méretét egy bemeneti forrásból (input), meghívja az *allocateMemoryForArray2(*pn, dpArray)*függvényt, amely segítségével dinamikusan lefoglal egy tömböt, majd feltölti azt a fájlból beolvasott adatokkal.

Megjegyzések:

- időbeli komplexitás **O(n)**
- térbeli komplexitás **O(n)**

Deklaráció:

```
void printArray(int n, int *pArray, const char *output);
```

Függvény dokumentáció:

```
/**
 * Kiírja egy n elemű tömb tartalmát egy adott kimeneti fájlba vagy a standard kimenetre.
 * @param n a tömb elemeinek száma.
 * @param pArray pointer egy tömb első elemére.
 * @param output karakterlánc, amely a kimeneti fájl nevét tartalmazza.
 */
```

Definíció:

```
void printArray(int n, int* pArray, const char* output)
{
    freopen(output, "w",stdout);
    for (int i = 0; i < n; i++)
    {
        printf("%i ", pArray[i]);
        //printf("%i ",*(pArray+i));
    }
    printf("\n");
    freopen("CON", "w",stdout);
}
```

Kódmagyarázat

A függvény egy n elemű tömb tartalmát írja ki egy fájlba vagy a képernyőre.

Megjegyzések:

- időbeli komplexitás **O(n)**
- térbeli komplexitás **O(n)**

Deklaráció:

```
void deallocateMemoryForArray(int **dpArray);
```

Függvény dokumentáció:

```
/**
 * Felszabadítja a dinamikusan lefoglalt memóriát
 * @param dpArray dupla pointer, amely a dinamikusan foglalt tömb első elemére mutat.
 */
```

Definíció:

```
void deallocateMemoryForArray(int** dpArray)
{
    free(*dpArray);
    *dpArray = NULL;
}
```

A függvény felszabadítja a memóriát, majd NULL-ra állítja a mutatót a további használati hibák elkerülése érdekében.

Megjegyzések:

- időbeli komplexitás **O(1)**
- térbeli komplexitás **O(1)**