Wstęp do programowania w języku C

Grupa MSz w czwartki

Lista 6 na zajęcia 24.11.2022

Zadanie 1. ***(10 punktów na pierwszej pracowni, 5 punktów na drugiej)***

Definiujemy strukturę `Array2D`, która interpretuje blok pamięci jako dwuwymiarową tablicę `int`'ów:

```
struct Array2D {
  unsigned int width, height;
  int *t;
};
```

Umówmy się, że int'y w bloku wskazywanym przez `t` definiują kolejne wartości w wierszu pierwszym, potem w drugim, itd. (albo kolumnami, do wyboru). Możemy też przyjąć, że wymiary będą zawsze dodatnie.

Zaimplementuj łatwe funkcje:

1. `int getValue(struct Array2D a, int x, int y);`

   Zwraca wartość w komórce o podanych indeksach.

2. `void setValue(struct Array2D a, int x, int y, int val);`

   Ustawia wartość w komórce o podanych indeksach.

3. `void print(struct Array2D a);`

   Ładnie wypisuje tablicę w dwóch wymiarach. Oczywiście użyj funkcji `getValue`.

4. `void fill(struct Array2D a, int val);`

   Wypełnia tablicę wartością `val`. Oczywiście użyj funkcji `setValue`.

A potem trudniejszą funkcję:

5. ```
   struct Array2D getSubarray(struct Array2D a,
       int left, int top, int right, int bottom);
   ```
   ```

Zwraca strukturę opisującą prostokątny fragment podanej tablicy. Oczywiście zakładamy, że wymiary mieszczą się w podanej tablicy (indeksujemy od 0). Współrzędne granic fragmentu są podane włącznie, czyli szerokość wynosi right−left+1. Aby to zrobić efektywnie (tj. bez kopiowania wartości, pomocniczych tablic itp.), trzeba najpierw rozszerzyć definicję struktury o pamiętanie oryginalnej szerokości tablicy.

Test:

```
int area[100];
struct Array2D a = {10,10,10,area};

struct Array2D s = a;

for (int i = 0; i < 5; i++) {
  fill(s, i);
  s = getSubarray(s, 1, 1, s.width-2, s.height-2);
}
print(a);
```

Zadanie 2. ***(10 punktów)***
To zadanie ma termin na zajęciach 8.12.2022.

Zaimplementuj funkcje:

```
struct Array2D newArray(unsigned int width, unsigned int height);
void freeArray(struct Array2D a);
```

które odpowiednio rezerwują i zwalniają pamięć dla tablicy (wywołanie malloc i free). (Można przy okazji wrócić do oryginalnej definicji struktury, pamiętającej tylko szerokość i wysokość.)

A potem:

```
resize(struct Array2D *a, unsigned int width, unsigned int height);
```

która zmienia rozmiary podanej tablicy w taki sposób, że:

- Komórki, nadal które mieszczą się w nowych wymiarach zachowują swoją wartość.
- Jeśli liczba kolumn się zwiększa, to nowe kolumny mają te same wartości co poprzednio ostatnia kolumna.
- Analogicznie dla wierszy.
- Aby zapewnić efektywność, funkcja wykonuje dokładnie jedno wywołanie `realloc` i nie używa żadnych pomocniczych tablic (działa w stałej pamięci).[1]

Przykład i test jednocześnie:

```
struct Array2D a = newArray(3,3);
for (int i = 0; i < 9; i++) a.t[i] = i+1;

print(a);
// 1 2 3
// 4 5 6
// 7 8 9

resize(&a,4,3); print(a);
// 1 2 3 3
// 4 5 6 6
// 7 8 9 9

resize(&a,5,2); print(a);
// 1 2 3 3 3
// 4 5 6 6 6
```

[1] Można zrobić bez tego punktu za połowę punktów.

```
resize(&a,2,3); print(a);
// 1 2
// 4 5
// 4 5

resize(&a,4,5); print(a);
// 1 2 2 2
// 4 5 5 5
// 4 5 5 5
// 4 5 5 5
// 4 5 5 5

free(a);
```

Zadanie 3. *Przypominam, że do każdej listy w SKOSie jest jeszcze do zrobienia zadanie dla sprawdzaczki, które ma osobny termin.*