Wojciech Myszka

Laboratorium 9a: Tablice dynamiczne

2020-04-26 12:46:26 +0200

1. Wprowadzenie

Zasadniczym tematem zajęć jest doskonalenie umiejętności operowania na **jednowymiarowych** tablicach dynamicznych (oraz używaniu ich jako parametrów funkcji).

Tematyka zadań nawiązuje do wcześniej rozpatrywanego problemu wyszukiwania binarnego. Tym razem program jest znacznie bardziej rozbudowany (wychodząc z założenia, że wszyscy ogarnęli problem odpowiedzi na pytanie *Gdzie należy wstawić szukaną wartość, gdy program nie znajduje jej w tablicy?*).

Wyszukiwaniu binarnemu poświęciłem sporo miejsca:

1. Laboratorium 5: Tablice. Wyszukiwanie binarne
2. Laboratorium 5a: Wyszukiwanie binarne — doskonalenie aplikacji
3. Tablice w czasach zarazy
4. Wyszukiwanie binarne dla bystrzaków

więc nie ma chyba sensu do problemu wracać ponownie.

2. Wyszukiwanie binarne z wstawianiem

Natomiast omówienia wymaga nowe postawienie zadania:

1. Program będzie czytał[1] kolejne wyszukiwane wartości z klawiatury.
2. Program startuje z **pustą** tablicą (która będzie rozszerzana dynamicznie):

```
int * dane;
int N = 0;
```

3. Po przeczytaniu pierwszej wartości (`x`) program wykona następujące czynności:

[1] Tak. Tym razem czytamy kolejne wartości z klawiatury!

```
dane = malloc(1 * sizeof(int))
dane[N] = x;
N++;
```

Teraz tablica ma 1 element.

4. Dla każdej kolejnej czytanej wartości, program sprawdza, czy znajduje się ona w tablicy:
   — jeżeli tak — nie robi nic, przechodzi do czytania kolejnej wartości;
   — jeżeli nie — zwiększa długość tablicy o 1
   `dane = realloc((N + 1) * sizeof(int));`
   — przestawia tak elementy tablicy żeby na odpowiedniej pozycji pojawiło się miejsce na nowy element;
   — wstawia tam `x` i zwiększa `N` o jeden
5. Po każdym dodaniu, dla kontroli, program drukuje aktualną zawartość tablicy.

2.1. Podział programu na funkcje

Proponuję następujący podział programu na funkcje:

1. Wyszukiwanie binarne.
2. Drukowanie tablicy.
3. Dodawanie elementu do tablicy (parametrami funkcji powinna być wstawiana wartość `x`, numer miejsca, na którym ma być wstawiona: 0 — na początku, `N` na końcu, rozmiar tablicy, tablica (adres jej początku))[2].

3. Parę wyjaśnień

Funkcja `malloc()` oraz spokrewnione z nią `calloc()`, `realloc()`, `reallocarray()` oraz `free()` służą do zarządzania pamięcią dostępną dla programu. Żeby z nich korzystać należy dołączyć do programu plik `stdlib.h` używając polecenia

```
#include <stdlib.h>
```

3.1. Funkcja `malloc()`

Funkcja ta zwraca się do Systemu Operacyjnego z prośbą o przydział pamięci. Jej argument to **liczba bajtów** przydzielanej pamięci. Zwracam uwagę, że we współczesnych komputerach największa liczba całkowita 32-bitowa to 2147483647 (czyli 2 G); biorąc pod uwagę, że każda zmienna typu `int` zajmuje 4 bajty to tablica może mieć co najwyżej 536870912, sporo, ale komputer może mieć znacznie więcej niż 2 G bajtów pamięci. Z tego względu argument funkcji `malloc()` jest specjalnego

[2] Przed wywołaniem tej funkcji tablica musi być już powiększona.

typu `size_t`. W pewnym uproszczeniu jest to typ całkowity, bez znaku, zapisywany na 8 bitach. Trzeba o tym pamiętać gdy potrzebujemy baaardzo dużo pamięci.

Pamięć przydzielane z użyciem tej funkcji nie jest w żaden sposób kasowana — otrzymamy obszar o niezdefiniowanej, „przypadkowej" zawartości.

Gdy pamięć nie może być przydzielona funkcja zwraca wartość `NULL` (zero).

Zatem zamiast używać polecenia:

```
int tablica[1000];
```

używać możemy:

```
int * tablica;
tablica = malloc(1000 * sizeof(int));
if (tablica == NULL)
    // błąd
    return 1;
```

3.2. Funkcja `calloc()`

Funkcja ta ma dwa argumenty (typu `size_t`): `nmemb` i `size`. Służy do przydzielenia **wyzerowanego** obszaru pamięci dla tablicy o `nmemb` elementach, zakładając, że każdy element zajmuje `size` bajtów.

Jeżeli chcemy stworzyć tablicę `T` typu double, o 1000 elementach powinniśmy użyć polecenia następującego:

```
double * T;
T = calloc(1000, sizeof(double));
```

(pierwsza wartość jest typu `int` ale zostanie automatycznie skonwertowana do typu `size_t`).

Gdy pamięć nie może być przydzielona funkcja zwraca wartość `NULL` (zero).

Zatem zamiast używać polecenia:

```
double Tablica[1000]={0.};
```

możemy napisać:

```
double * Tablica;
Tablica = calloc(1000, sizeof(double));
```

3.3. Funkcja `realloc()`

Funkcja ta, o dwu parametrach:

— pierwszy to, adres początku tablicy utworzonej za pomocą `malloc()`,
— drugi to nowy rozmiar pamięci (w bajtach)

zmienia przydział pamięci powiększając lub zmniejszając obszar tablicy. Dotychczasowa zawartość obszaru pamięci pozostanie niezmieniona, gdy tablica jest rozszerzana, dodawana pamięci nie będzie zmieniana w żaden sposób (czyli pojawią się tam jakieś „śmieci"). Funkcja zwraca adres obszaru pamięci o nowym rozmiarze lub wartość `NULL` gdy nie może zadania wykonać. Trzeba o tym pamiętać podczas zwiększania rozmiaru tablicy:

```
double * T;
T = malloc(N * sizeof(double));
// dalszy ciąg programu, zmienia się wartość N
T = realloc(T, N * sizeof(double));
```

gdy żądanie nie będzie mogło być zrealizowane, `T` przyjmie wartość zero i **stracimy** dostęp do poprzedniej wartości tablicy `T`. Czyli powinno to być raczej robione inaczej

```
double * Tprim;
Tprim = realloc(T, N * sizeof(double));
if ( Tprim != NULL)
    T = Tprim;
```

3.4. Funkcja `reallocarray()`

Funkcja ta to odpowiednik funkcji `realloc()` ale operujący liczbą (i wielkością) elementów tablicy, podobnie jak funkcja `calloc()`. Jej parametry to:

— dotychczasowy adres tablicy,
— nowa liczba elementów tablicy,
— wielkość jednego elementu w bajtach.

Mają zastosowanie wszystkie uwagi dotyczące funkcji `realloc()`

3.5. Funkcja `free()`

Funkcja ta zwalnia obszar pamięci przydzielony za pomocą `malloc()`. Jej argumentem jest wskaźnik (adres) używanej tablicy. Używając tej funkcji nie można zwolnić pamięci uzyskanej za pomocą „normalnej" deklaracji zmiennej.

Funkcja nie zwraca żadnej wartości

3.6. Prototypy funkcji

```
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

```
void *reallocarray(void *ptr, size_t nmemb, size_t size);
```

Wszystkie funkcje zwracające wskaźnik zwracają wskaźnik **bez określonego typu** (`void *`). Podczas podstawienia dokonywane jest odpowiednie rzutowanie.

4. Wersja PDF tego dokumentu...

...pod adresem.

Wersja: 57 z drobnymi modyfikacjami! data ostatniej modyfikacji 2020-04-26 12:46:26 +0200