# 12. Sortowanie

## 1 Konstrukcja posortowanej tablicy elementów unikalnych z wykorzystaniem zmodyfikowanej funkcji `bsearch`

### 1.1 O zmodyfikowanej funkcji `bsearch()`:

Oryginalna, biblioteczna funkcja `bsearch` zwraca adres znalezionego elementu tablicy albo – w przypadku braku szukanego elementu – `NULL`. Modyfikacja ma zastosowanie w sytuacji gdy szukany element (gdy nie został znaleziony) ma być dodany do tablicy z zachowaniem obowiązującego w tablicy porządku. Funkcja `bsearch` "dociera" do informacji, pod jakim adresem "powinien" być szukany element (ale go tam nie ma). Jednak informacja ta jest tracona. Modyfikacja polega na przekazaniu (do funkcji wywołującej) adresu, pod którym należałoby umieścić szukany element.

### 1.2 Zadanie

Szablon programu należy uzupełnić o definicję funkcji `void *bsearch2(const void *key, const void *base, const size_t nitems, const size_t size, const CompareFp compare, char *result)`.

Możliwe są dwa rezultaty szukania:

1. Sukces szukania – funkcja wpisuje pod adres `result` wartość różną od zera oraz zwraca adres znalezionego elementu (jak oryginalna funkcja `bsearch`).
2. Element z kluczem `*key` nie został znaleziony – funkcja wpisuje zero pod adres `result` oraz zwraca adres, pod który należy wpisać nowy element zachowując przyjęty porządek. Funkcja nie sprawdza, czy zwracany adres nie przekracza zakresu pamięci przydzielonej tablicy `base`.

Zadanie to jest przykładem zastosowania funkcji `bsearch2`. Obejmuje wczytanie danych o artykułach spożywczych (cena jednostkowa, liczba sztuk, termin ważności[1] `dd.mm.yyyy`, nazwa) i zapisywanie ich w określonym porządku w tablicy struktur typu `Food`. Deklaracja tej struktury, zawierającej dane jednej partii artykułu, jest zapisana w szablonie.

Jeżeli wczytany rekord zawiera dane o artykule, który jest już zapisany w tablicy oraz dane wczytane różnią się (lub nie) od zapisanych tylko wartością w polu `amount`, to nie należy tworzyć

[1] "Termin ważności" należy traktować jako skrótowe określenie np. terminu przydatności do spożycia

dla tego rekordu nowego elementu tablicy, lecz zwiększyć wartość w polu `amount` istniejącej już struktury.

Zapisywanie w tablicy struktur odczytanego ze strumienia wejściowego rekordu zawierającego dane o jednej partii artykułu ma zachowywać kolejność ustaloną wg kryteriów. Należy określić taką relację porządkującą elementy tablicy, aby odszukanie elementu, którego wartość pola `amount` ma być powiększona, wymagało tylko jednokrotnego wywołania funkcji `bsearch2`.

Kolejność elementów wg zadanego porządku ma być zachowana także w trakcie wpisywania nowych danych do tablicy.

Oprócz opisanej wyżej funkcji `bsearch2()`, szablon programu należy uzupełnić o:

1. Definicję funkcji `Food *add_record(Food *tab, int *np, CompareFp compare, const Food *new)`, która wywołuje funkcję `bsearch2()` sprawdzającą, czy nowy artykuł (jego dane są zapisane pod adresem `*new`) jest zapisany w tablicy `tab` o `*np` elementach. O tym, czy uznać `*new` za nowy decyduje funkcja wskazywana pointerem do funkcji `compare` (typu `CompareFP` – zdefiniowanego w szablonie).
   - Jeżeli `*new` nie jest elementem nowym, to dane zapisane w elemencie tablicy są modyfikowane danymi zapisanymi w `*new`. Konkretnie, ilość artykułu znalezionego w tablicy jest powiększana o ilość zapisaną w `*new`. Funkcja zwraca adres modyfikowanego elementu tablicy.
   - Jeżeli `*new` jest elementem nowym, to funkcja `add_record` dodaje we wskazanym miejscu nowy element (z ewentualnym przesunięciem części elementów tablicy), zwiększa liczbę elementów tablicy `*np` i zwraca adres wpisanego elementu.
2. Definicję funkcji wskazywanej pointerem `compare`.
3. Definicję funkcji `int read_goods(Food *tab, FILE *stream, const int sorted)`, która czyta liczbę linii danych (`no`) a następnie `no` linii ze strumienia wejściowego. W tym przypadku parameter `sorted` powinien mieć wartość 1, co oznacza, że dla każdego rekordu wywołuje funkcję `add_record` zachowującą porządek sortowania. Funkcja zwraca aktualną długość tablicy `tab` po wczytaniu wszystkich danych.

**Test 1**

Test wczytuje liczbę wprowadzanych linii danych, wywołuje funkcję `read_goods()`, wczytuje nazwę artykułu i wypisuje wszystkie dane zawarte w strukturach z wskazaną nazwą artykułu w kolejności: po pierwsze – rosnącej ceny, w drugiej kolejności – "rosnącego" terminu ważności.

- **Wejście**
  1
  liczba linii n
  n linii: nazwa cena ilość dd.mm.yyyy
  nazwa artykułu
- **Wyjście**
  pamiętane w tablicy dane o artykule o wczytanej nazwie

cena ilość dd.mm.yyyy
cena ilość dd.mm.yyyy
...

- **Przykład**
  Wejście: 1
  6
  kefir 3.50 30 7.6.2023
  ser 7.80 25 15.6.2023
  kefir 3.75 20 7.6.2023
  ser 7.80 12 15.6.2023
  mleko 3.25 44 29.12.2023
  kefir 3.50 22 7.6.2023
  kefir

  Wyjście:
  3.50 52 07.06.2023
  3.75 20 07.06.2023

## 2 Sortowanie elementów tablicy struktur

Zadanie polega na posortowaniu biblioteczną funkcją `qsort` tablicy struktur utworzonej w zadaniu 1. Relację porządkującą elementy tablicy należy zdefiniować tak, aby przy możliwie małym koszcie obliczeniowym (dla dużej liczby danych) obliczyć wartość towaru, którego termin ważności mija dokładnie po $d$ dniach od założonej daty (termin ważności = $d$ dni + zadana data). Zadana data symuluje tu datę bieżącą.

Sugestia wyboru algorytmu: Posortować (`qsort`) wg daty, odszukać (bsearch) jeden element - w jego bezpośrednim "sąsiedztwie" są pozostałe z szukaną datą.

Należy zwrócić uwagę na okresy przełomu miesięcy lub lat. Zadanie można sobie ułatwić korzystając z funkcji deklarowanych w pliku nagłówkowym `time.h` standardowej biblioteki (funkcja `mktime`).

Szablon programu należy uzupełnić o:

1. Do odczytu danych można wykorzystać funkcję `read_goods()` z poprzedniego punktu z wartością parametru `sorted` = 0.
2. Definicję funkcji `float value(Food *food_tab, const size_t n, const Date curr_date, const int days)`, która oblicza omawianą wyżej wartość artykułów.

**Test 2**

Test wczytuje dane tak, jak w zadaniu 1 (liczbę rekordów danych, rekordy danych, zadaną datę i liczbę dni do sprawdzanej daty ważności). Następnie wywołuje funkcję `value()`, która oblicza sumę wartości wszystkich artykułów, które tracą ważność za $d$ dni (przykład: jeżeli 5.6.2023 będzie wczytaną datą, a $d = 5$, to będą poszukiwane artykuły o terminie ważności 10.6.2023).

- **Wejście**
  2
  liczba linii $n$
  $n$ linii: nazwa cena ilość `dd.mm.yyyy`
  data (symulująca datę bieżącą)
  liczba $d$ dni do szukanej daty ważności.

- **Wyjście**
  Suma wartości artykułów z wskazaną datą ważności

- **Przykład:**
  Wejście:
  2
  6
  kefir 3.50 30 7.6.2023
  ser 7.80 25 15.6.2023
  kefir 3.75 20 7.6.2023
  ser 7.80 12 15.6.2023
  mleko 3.25 44 29.12.2023
  kefir 3.50 22 7.6.2023
  2 6 2023
  5

  Wyjście:
  257.00