10 Operacje na uogólnionych wektorach

Uwagi ogólne

Celem ćwiczenia jest zaimplementowanie struktury opisującej wektor elementów dowolnego typu oraz funkcji realizujących operacje na takim wektorze. Dobór i nazewnictwo funkcji jest inspirowane klasą `vector` z biblioteki STL C++. Z oczywistych względów realizacja jest zupełnie inna.

W kolejnych punktach na podstawie stworzonego szablonu będziemy tworzyć wektory liczb całkowitych, znaków, oraz struktur `Person`.

Struktura wektor i funkcje ogólne

Struktura `Vector` składa się z:

1. wskaźnika do początku (dynamicznie alokowanej) tablicy przechowującej dane wektora
2. rozmiaru elementu wektora (w bajtach)
3. aktualnej liczby elementów wektora
4. pojemności wektora (rozmiaru aktualnie zaalokowanej tablicy danych)

Szablon programu należy uzupełnić o definicję następujących funkcji:

1. `init_vector()` – alokuje tablicę danych na zadaną pojemność początkową i inicjalizuje pozostałe pola.
2. `reserve()` – realokuje tablicę danych tak, żeby miała co najmniej zadaną pojemność. Jeżeli zadana pojemność nie przekracza aktualnej funkcja nic nie robi.
3. `resize()` – zmienia aktualną liczbę elementów wektora: jeżeli nowy rozmiar jest mniejszy od aktualnego, nadmiarowe elementy są usuwane; jeżeli nowy rozmiar jest większy, wektor jest uzupełniany o odpowiednią liczbę wyzerowanych elementów.
4. `push_back()` – dodaj element na koniec wektora.
5. `insert()` – dodaj element na zadanej pozycji.
6. `clear()` – usuń wszystkie elementy z wektora.
7. `erase()` – usuń element wektora na zadanej pozycji.
8. `erase_value()` – usuń wszystkie elementy wektora o zadanej wartości.
9. `erase_if()` – usuń wszystkie elementy wektora spełniające predykat.
10. `shrink_to_fit()` – dopasuj rozmiar tablicy do aktualnej liczby elementów wektora
11. `print_vector()` – wypisz pojemność wektora i jego elementy

Pomocne uwagi:

1. Do zmiany wielkości tablicy (pojemności wektora) proszę używać funkcji `realloc()`
2. Przy dodawaniu elementów do wektora (funkcje `push_back(), insert()`) jeżeli konieczne jest zwiększenie jego pojemności, pojemność jest zwiększana dwukrotnie i tablica danych jest realokowana.
3. Do kopiowania fragmentów pamięci o zadanym rozmiarze proszę wykorzystać funkcję `memcpy()` (jeżeli obszary nie mają części wspólnej) lub `memmove()` (jeżeli się przecinają).

Ogólna postać danych (do każdego podpunktu):

numer zadania

`n` – liczba komend

`n` linii komend

Każda komenda składa się z litery (kodu komendy) i pozostałych danych (w zależności od typu polecenia).
Lista komend:

1. `p value - push_back(value)`
2. `i index value - insert(index, value)`
3. `e index - erase(index)`
4. `v value - erase_value(value)`
5. `d - erase_if()` – predykat definiowany jest dla konkretnego typu
6. `r new_size - resize(new_size)`
7. `c - clear()`
8. `f - shrink_to_fit()`
9. `s - qsort()`

Wyjściem z każdej sekcji jest pojemność i elementy wektora po wykonaniu wszystkich operacji.

10.1 Wektor liczb całkowitych

Dodatkowo szablon programu należy uzupełnić o funkcje:

1. `read_int()` – czytaj wartość całkowitą do adresu wskazywanego przez `value`
2. `print_int()` – wypisz wartość całkowitą
3. `int_cmp()` – komparator wartości całkowitych (sortowanie malejące)
4. `is_even()` – predykat (zwraca 1 jeżeli liczba jest parzysta).

Przykład:
Wejście:

```
1
14
p 10
p 20
p 5
p 3
p 15
i 0 30
i 4 40
i 7 50
e 4
v 10
e 4
v 20
r 6
f
```

Wyjście:

```
6
30 5 3 50 0 0
```

10.2 Wektor znaków

Dodatkowo szablon programu należy uzupełnić o funkcje:

1. `read_char()` – czytaj wartość typu `char` do adresu wskazywanego przez `value`
2. `print_char()` – wypisz wartość typu `char`
3. `char_cmp()` – komparator wartości znakowych (porządek leksykograficzny)
4. `is_vowel()` – predykat (zwraca 1 jeżeli znak jest samogłoską)

Przykład:
Wejście:

```
2
10
p a
p X
p k
p i
p y
p R
i 1 t
i 3 G
i 5 E
d
```

Wyjście:

```
16
t X G k R
```

10.3 Wektor struktur `Person`

Dodatkowo szablon programu należy uzupełnić o funkcje:

1. `read_person()` – czytaj elementy struktury `Person` do adresu wskazywanego przez `value`
2. `print_person()` – wypisz strukturę `Person`
3. `person_cmp()` – komparator struktur `Person` (malejąco wg wieku, następnie imienia i nazwiska – rosnąco)
4. `is_older_than_25()` – predykat (zwraca 1 jeżeli osoba ma więcej niż 25 lat)

Przykład:
Wejście:

```
3
8
p 23 Dominik Adamczyk
p 27 Natalia Adamiak
p 24 Marcin Chudy
i 1 29 Anna Cichocka
i 4 22 Natalia Deyna
i 0 24 Marcin Bereta
d
s
```

Wyjście:

```
8
24 Marcin Bereta
24 Marcin Chudy
23 Dominik Adamczyk
22 Natalia Deyna
```