Programowanie 1

Sławomir Pluciński (splucinski@pjwstk.edu.pl)

Zapis i odczyt danych z pliku

Obsługa pliku

Do obsługi operacji pozwalających na zapis i odczyt danych do pliku odpowiada biblioteka **fstream.**

Dodając wskazaną bibliotekę uzyskujemy dostęp do klasy std::fstream. Klasa ta udostępnia interfejs do obsługi dowolnego pliku na dysku komputera.

Pierwszym wymaganiem jest utworzenie zmiennej typu fstream, która będzie wskazywać na obsługiwany plik.

```
std::fstream plik
```

Otwarcie pliku

Należy wprost wskazać jaki plik i w jakim trybie powinien zostać otwarty. Służy do tego funkcja:

```
void open( const char *filename,
           ios_base::openmode mode = ios_base::in|ios_base::out );
```

Tryb	Opis trybu
ios::app	(**app**end - dopisywanie danych do pliku) Ustawia wewnętrzny wskaźnik zapisu pliku na jego koniec. Plik otwarty w trybie tylko do zapisu. Dane mogą być zapisywane tylko i wyłącznie na końcu pliku.
ios::ate	(**at** end) Ustawia wewnętrzny wskaźnik pliku na jego koniec w chwili otwarcia pliku.
ios::binary	(**binary**) Informacja dla kompilatora, aby dane były traktowane jako strumień danych binarnych, a nie jako strumień danych tekstowych.
ios::in	(**in**put - wejście/odczyt) Zezwolenie na odczytywanie danych z pliku.
ios::out	(**out**put - wyjście/zapis) Zezwolenie na zapisywanie danych do pliku.
ios::trunc	(**trunc**ate) Zawartość pliku jest tracona, plik jest obcinany do 0 bajtów podczas otwierania.

Sprawdzenie poprawności otwarcia pliku

Interfejs dostarcza funkcje sprawdzenie, czy udało się poprawnie uzyskać dostęp do pliku oraz czy plik jest otwarty:

```
std::fstream plik

plik.good() //return true / false
plik.is_open() //return true / false
```

Zamykanie pliku

Przy pracy na plikach wskazane jest jego zamknięcie po zakończeniu pracy z plikiem. Jeżeli zapomnisz zamknąć plik klasa fstream zrobi to sama przed usunięciem zmiennej z pamięci.

Zamykanie realizowane jest przez funkcję:

```
std::fstream plik

plik.close() //return void
```

Odczytanie danych z pliku

Aby odczytać dane z pliku mamy do dyspozycji kilka możliwości.

Najprostszym sposobem jest wykorzystanie strumienia i podobnie jak z wykorzystaniem cin >>

```
std::fstream plik
string dane;

plik >> dane
```

- Dane w tej sposób odczytane zawsze są traktowane jako tekst
- Działa tak samo jak cin, więc pobierany napis nie będzie zawierał białych znaków (enter, spacja, tab)

Odczytanie danych z pliku

Alternatywnie stosować można funkcję:

```
fstream plik( "plik.txt", ios::in );
string linia;
getline( plik, dane );
```

Która zwróci zawartość całej jednej linii i zapisze do zmiennej wskazanej jako parametr. Funkcja zwraca jako typ bool, przez co dowiadujemy się kiedy nie ma już nie ma linii do odczytu.

Dane pobierane są traktowane jako tekst niezależnie czy wskażemy trym odczytu jako ios::binary.

Odczytanie danych z pliku

Dostępna jest również funkcja getline() z klasy **fstream.**

```
istream& getline(char* dane, streamsize wielkosc_danych, char znak_konca);
```

Parametry oznaczają kolejno:

(dane) wskaźnik na zmienna do której mają zostać zapisane dane.

(wielkość_danych) ilość danych, która zostanie zapisana.

(znak_konca) Wskazuje znak końca linii (opcjonalny).

```
fstream plik( "plik.txt", ios::in );
char linia[512];
plik.getline(linia, 512, '$' );
```

Odczytanie danych z pliku

Pobieranie danych blokami umożliwia pobieranie danych binarnych. (Jeżeli zadeklarowaliśmy tryb ios::binary):

```
istream& read( char* bufor, streamsize rozmiar_bufora );
```

Tak będzie wyglądało wywołanie tej funkcji:

```
fstream plik( "plik.txt", ios::in );
char bufor[512]
plik.read( bufor, 512);
```

To ile danych zostało wczytanych możemy sprawdzić za pomocą metody **gcount().**

Zapis danych do pliku

Najprostszym sposobem do zapisu danych do pliku jest wykorzystanie strumieni, analogicznie jak cout <<.

```
fstream plik( "plik.txt", ios::in | ios::out );
plik << "Zapisane dane"
```

Dane zostaną zapisane jako tekst, niezależnie od trybu otwarcia pliku. Zapisanie danych powoduje przesunięcie wskaźnika o tyle znaków ile zostało zapisane do pliku.

Zapis danych do pliku

Dla zapisu danych w postaci innej niż tekstowa wykorzystywać będziemy funckję write() z klasy **fstream**.

```
ostream& write(const char* bufor, streamsize ilosc_danych);
```

Parametry oznaczają kolejno:

(bufor) wskaźnik na bufor w którym znajdują się dane do zapisu.

(ilosc_danych) ilość danych, która zostanie zapisana.

```
fstream plik("plik.txt", ios::out);
string napis = "Napis";
plik.write(&napis[0], napis.length());
```

Odczytanie całego pliku

Jeżeli przyjdzie nam przeczytać cały plik to pomocną będzie funkcja eof() z klasy fstream. Metoda ta informuje, czy dotarła do końca pliku.

Przykład z zastosowaniem:

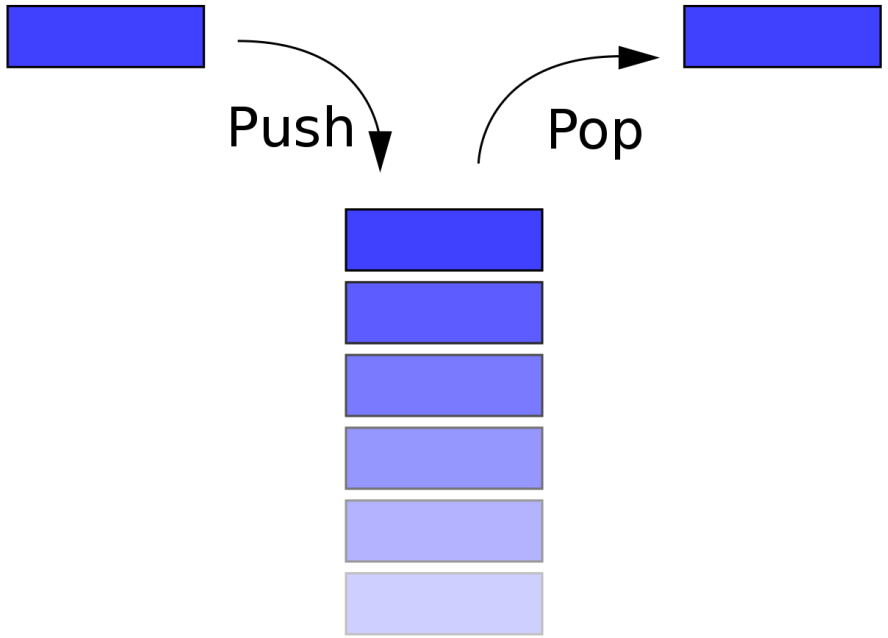
```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main() {

    fstream file("file.txt", ios::in | ios::out );

    if(file.good()){
        string line;
        while(!file.eof()){
            getline(file, line);
            cout << line << endl;
        }
        file.close();
    }
}
```

Struktury danych

Stos

Definicja

Liniowa struktura danych – dane ustawiane są na wierzchu stosu i z wierzchołka są pobierane. Sposób działania tej struktury opisuje zasada LIFO (last in first out). Dla użytkownika stosu dostępna jest tylko jedna wartość na jego wierzchołku.



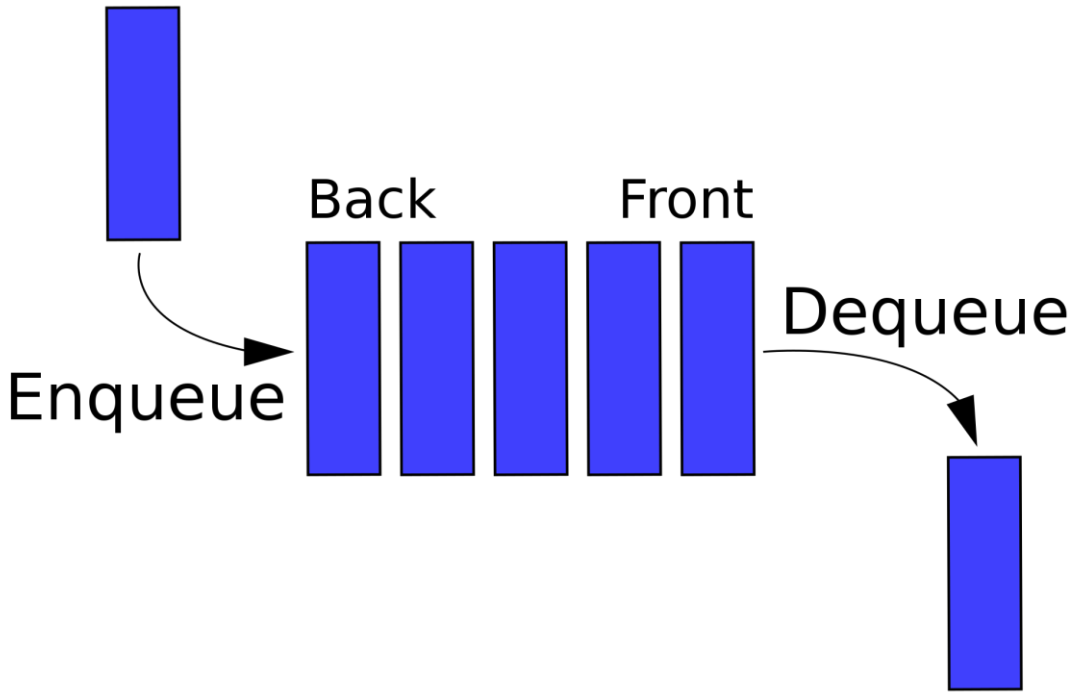
Dostępne operacje

- push() – dodanie elementu na stos
- pop() – pobranie elementu ze stosu
- size() – liczba elementów na stosie
- empty() – informacja, czy stos jest pusty

Kolejka

Definicja

Liniowa struktura danych – dane dodawane są do końca kolejki (ogona), a pobierane są z początku (głowy) kolejki. Takie podejście nazywamy FIFO (first in first out).



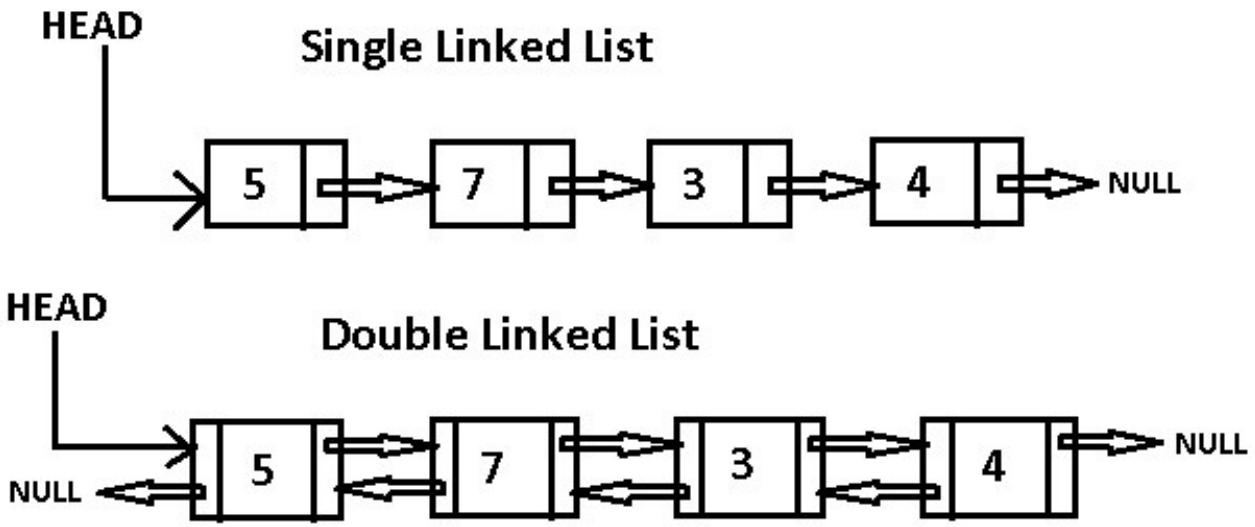
Dostępne operacje

- push() – dodanie elementu na koniec kolejki
- pop() – pobranie pierwszego elementu kolejki
- size() – liczba elementów w kolejce
- empty() – informacja, czy kolejka jest pusta

Lista

Definicja

Struktura danych wspierająca dodawanie i usuwanie elementów z dowolnego miejsca w kontenerze. Zwykle implementowana jako listwa dwukierunkowa, która posiada nie tylko przetrzymywaną wartość ale również wskaźnik na element poprzedni i element następny. Przechowuje dane tego samego typu.



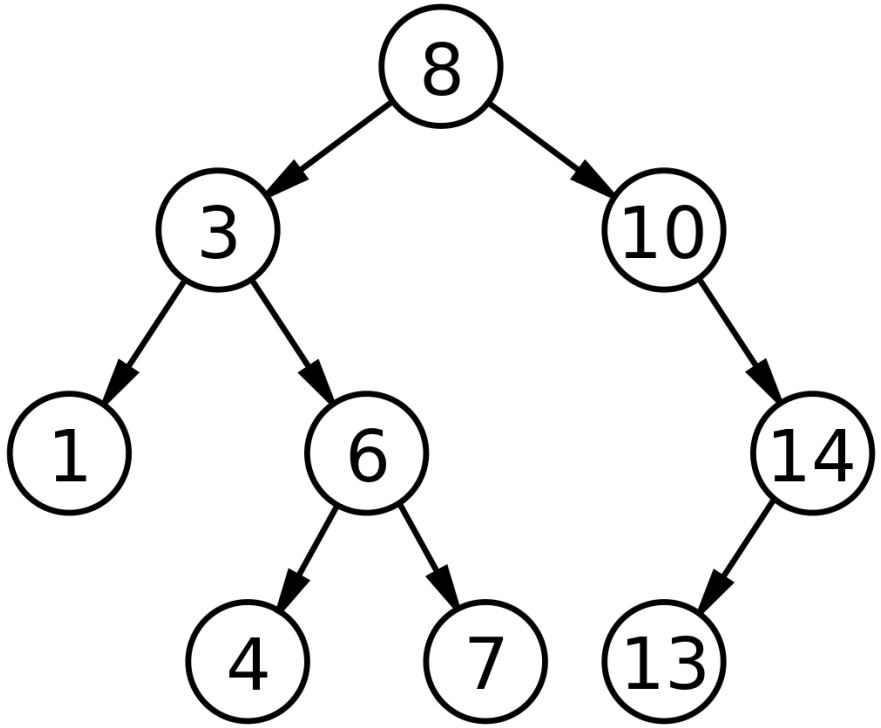
Dostępne operacje

- push_front() – dodanie elementu na początek listy
- push_back() – dodanie elementu na koniec listy
- insert() – dodanie elementu we wskazanym miejscu listy
- pop_front() – pobranie elementu z początku listy
- pop_back() – pobranie elementu z końca listy
- size() – liczba elementów w liście.
- max_size() – maksymalna liczba elementów w liście.
- empty() - informacja, czy lista jest pusta
- remove() – usuwa wszystkie wystąpienia wskazanej wartości
- sort() – sortuje elementy rosnąco
- reverse() – odwraca kolejność elementów na liście

Drzewo binarne

Definicja

Hierarchiczna struktura, gdzie wyróżniamy węzeł lub wierzchołek od którego tworzone są nie więcej niż dwa potomki (następniki). Pierwszy element drzewa nazywamy korzeniem. Dodając elementy do drzewa należy stosować zasadę, że mniejsze elementy ustawiamy na lewo od wierzchołka, a większe na prawo od wierzchołka.



Dostępne operacje

- add() – dodanie elementu do drzewa
- find() – odnalezienie elementu w drzewie
- remove() – usuwanie elementu z drzewa
- print() – prezentowanie zawartości drzewa