Strumienie

- strumień jest obiektem który formatuje i przechowuje bajty
- „opakowuje” komunikację z peryferiami
- komunikacja odbywa się przy pomocy stringów
- strumień wyjściowy zamienia przekazane mu dane, zmienne, stringi, obiekty, wskaźniki itp, na stringi i wysyła je do urządzenia wyjściowego
- strumień wejściowy odczytuje bajty z urządzenia wejściowego, odpowiednio je formatuje i zwraca
- do przekazywania danych służą operatory >> i <<

Strumienie

- standardowo zdefiniowane są strumienie (klasy) do komunikacji z konsolą/klawiaturą (`iostream`), z plikami (`iofstream`), z pamięcią (`iostrstream`) i ze stringami (`iostringstream`)
- standardowo zdefiniowane są też strumienie `cin` i `cout`
- na podstawie standardowych klas definiuje się własne klasy, na przykład strumień do komunikacji z czujnikiem temperatury itp
- manipulatory: `endl`, `hex`, `ws`, `flush`

Strumienie

- `getline()`, `get()` - metody strumienia wejściowego - pobieranie kawałków danych ze strumienia, typowo całej linijki, bez formatowania - pozwalają uniknąć problemów w przypadku niewłaściwych danych wejściowych
- `read()`, `write()` pozwalają manipulować poszczególnymi bajtami
- status strumienia: `good()`, `eof()`, `fail()`, `bad()`
- bity statusu: `goodbit`, `eofbit`, `failbit`, `badbit` można kasować: funkcja `ios::clear()`
- można stosować konstrukcje `while(cin)` itp

prostyplik.cpp

```
#include<iostream>
#include<fstream>
using namespace std;

int main()
{
    const int sz = 100;
    char buf[sz];
    {
        ifstream in("prostyplik.cpp");
        ofstream out("prostyplik.out");
        int i = 1;  // licznik linii
```

prostyplik.cpp

```
        while(in.get(buf, sz))
        {
            in.get();
            cout << buf << endl;
            out << i++ << ": " << buf << endl;
        }
    }   // destruktory obiektów in i out pilnują
        // zamknięcia plików
```

prostyplik.cpp

```
    ifstream in("prostyplik.out");
    while(in.getline(buf, sz))
    {
        char *cp = buf;
        while(*cp != ':')
            cp++;
        cp += 2;  // zjeść ": "
        cout << cp << endl;
    }
}
```

Strumienie

- tryb otwarcia plików można kontrolować: `ios::in`, `ios::out`, `ios::app`, `ios::ate`, `ios::nocreate`, `ios::noreplace`, `ios::trunc`, `ios::binary`
- głównym elementem strumienia jest bufor danych. Normalnie nie potrzebujemy „ręcznie” komunikować się z buforem
- czasem dobrze jest komunikować się z buforem bezpośrednio - bufor reprezentowany jest przez obiekt `streambuf`
- metoda `rdbuf()` strumienia zwraca wskaźnik do związanego z tym strumieniem `streambuf`

Przykład: wypisz1.cpp

Strumienie

- można posługiwać się buforem strumienia docelowego
- metoda `ignore()` odczytuje i pomija znaki - domyślnie jeden
- `in.ignore()` potrzebne jest do usunięcia znaków końca linii

Przykład: wypisz2.cpp

Przeszukiwanie strumienia

- strumień „wie” w jakim miejscu bufora obecnie się znajduje - gdzie trafi następny bajt lub skąd zostanie pobrany
- zmienna typu `streampos` służy do przechowywania pozycji w strumieniu. Zmienną tego typu zwracają metody `tellp()` (`ostream` - wskaźnik do put) i `tellg()` (`istream` - wskaźnik do get)
- zmienna `streampos` może być używana z jednoparametrowymi funkcjami `seekp()` i `seekg()` do ustawienia bieżącego miejsca w buforze (ustwienie absolutne)
- dwuparametrowe wersje powyższych funkcji służą do przejścia konkretnej ilości miejsc (pierwszy parametr - dodatni lub ujemny) od konkretnej pozycji w buforze (drugi parametr)
- możliwe pozycje: `ios::beg` - początek strumienia, `ios::cur` - bieżące miejsce w strumieniu, `ios::end` - koniec strumienia

Przykład: szukaj1.cpp

Przeszukiwanie strumienia

- plik może służyć jednocześnie do odczytu i zapisu
- wymaga to użycia bufora, kompilator nie pozwoli pisać do strumienia wejściowego
- używamy konstrukcji:

```
ifstream in("nazwa-pliku", ios::in|ios::out);
ostream out(in.rdbuf());
```

Przykład: szukaj2.cpp

Strumienie stringowe

- przy pomocy schematu strumienia możemy komunikować się ze stringami
- biblioteka `strstream` definiuje strumienie do komunikowania się z tablicami znaków bezpośrednio w pamięci - to są stringi starego typu, zakończone znakiem `'\0'`

```
istrstream::istrstream( char *buf);
istrstream::istrstream( char *buf, int size)
```

- pierwszy konstruktor oczekuje wskaźnika do tablicy znaków zakończonej znakiem `'\0'`
- drugi konstruktor nie potrzebuje znaku `'\0'` w tablicy - można ją odczytywać do `buf[size]`

Przykład: istring.cpp

Strumienie stringowe

- strumień wyjściowy korzysta z konstruktora:

```
ostrstream::ostrstream( char *,
                        int, int = ios::out);
```

- pierwszy parametr to wskaźnik do istniejącej tablicy znaków (trzeba zarezerwować pamięć wcześniej)
- drugi parametr to rozmiar tablicy
- trzeci parametr to tryb dostępu. Domyślnie zapisywanie zacznie się na początku bufora
- jeżeli ustalimy tryb na `ios::ate` lub `ios::app` to strumień wyszuka znak `'\0'` w buforze (koniec istniejącego stringu) i od tego miejsca dopisze
- strumień nie dopisuje automatycznie znaku `'\0'`. Trzeba to zrobić „ręcznie" aby zakończyć string. Służy do tego manipulator `ends`

Przykład: ostring.cpp