Strumienie

- włączając plik nagłówkowy `<iostream>` uzyskaliśmy dostęp do obiektów `cin` oraz `cout`
- obiekty te „opakowują” komunikację z konsolą
- komunikacja odbywa się przy pomocy stringów - wpisujemy w strumień string i strumień stara się to zinterpretować jako dane, których oczekuje
- drugą stronę też - dostajemy sformatowany string
- można też wpisywać dodatkowe instrukcje dla strumienia
- Przykład `prog1.cpp`

Strumienie

- strumień stanowi interfejs do ciągu bajtów
- taki ciąg bajtów może reprezentować konsolę, plik, string, jakiś zewnętrzny hardware
- w pliku `<iostream>` zawarte są strumienie związane z komunikacją z konsolą: `cin`, `cout`, `cerr`
- ze strumieniem związane są operatory >> i <<
- są to operatory *przeładowane*, pamiętamy ich działanie jako operatorów bitowych
- Przykład `prog2.cpp`

Strumienie

- oprócz `<iostream>` służącego do komunikacji z konsolą standardowo zdefiniowane są strumienie do komunikacji z plikami (`<iofstream>`), z pamięcią (`<iostrstream>`) i ze stringami (`<iostringstream>` to są stringi typu C++, omówimy je w następnej kolejności)
- strumienie te są klasami pochodnymi jednej wspólnej klasy, i użycie ich jest podobne: korzysta z operatorów >> i <<
- na podstawie standardowych klas definiuje się własne klasy, na przykład strumień do komunikacji z czujnikiem temperatury itp.
- strumień `cin` wraz ze swoim operatorem >> wygląda jak bardzo podobny do `cout`, ale w rzeczywistości nie jest często używany: źle wprowadzone dane dają nieprzewidziane rezultaty
- Przykład `prog3.cpp`

Strumienie

- `getline()`, `get()` - metody strumienia wejściowego - pobieranie kawałków danych ze strumienia, typowo całej linijki, bez formatowania - pozwalają uniknąć problemów
- funkcje są podobne, przyjmują 3 parametry: wskaźnik do bufora, pojemność bufora (limit bajtów wejściowych) oraz znak kończący wczytywanie (ostatni parametr jest opcjonalny, domyślnie tym znakiem jest `'\n'`)
- metoda `getline()` czyta do znaku kończącego, i pobiera go (i ignoruje)
- metoda `get()` czyta do znaku kończącego i zostawia go w strumieniu
- istnieją przeładowane wersje `get()` - jeżeli nie podamy żadnego argumentu, funkcja po prostu pobierze kolejny bajt ze strumienia, i zwraca go jako `int`
- Przykład `prog4.cpp`

Strumienie

- do pracy z plikami służą strumienie `ifstream` i `ofstream`, zawarte w bibliotece `<fstream>`
- obiekty `ifstream` związane są z plikami do odczytu, a obiekty `ofstream` związane są z plikami do zapisu

```
#include<fstream>
...
ifstream in("prostyplik.cpp");
ofstream out("prostyplik.out");
```

- status strumienia: metody `good()`, `eof()`, `fail()`, `bad()`
- bity statusu: `goodbit`, `eofbit`, `failbit`, `badbit` można kasować: funkcja `ios::clear()`
- można stosować konstrukcje `while(in.get())` itp
- Przykład `prog5.cpp`

Strumienie

- tryb otwarcia plików można kontrolować: `ios::in`, `ios::out`, `ios::app`, `ios::ate`, `ios::nocreate`, `ios::noreplace`, `ios::trunc`, `ios::binary` - jest to dodatkowy parametr konstruktora
- 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 `Prog6.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 `Prog7.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 `prog8.cpp`

Przeszukiwanie strumienia

- jeżeli plik chcemy otworzyć jednocześnie do odczytu i zapisu, korzystamy ze strumienia `fstream`
- jeżeli chcemy tylko czytać lub tylko pisać, używamy odpowiednio `ifstream` lub `ofstream` - te strumienie mają domyślnie ustawiony tryb dostępu, i kompilator nie pozwoli na inny tryb
- możliwy jest jednak dostęp w innym trybie, ale wymaga to użycia bufora `streambuf`
- używamy konstrukcji:

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

- Przykład `prog9.cpp`

Przeładowanie operatorów << >>

- przypomnijmy, strumień wyjściowy operuje na ciągu znaków, i formatują go. Zmienne różnych typów wypisywane są domyślnie w określony sposób
- liczby całkowite w systemie dziesiątkowym
- zmienne typu `char` wypisywane są jako znak o określonym kodzie ASCII
- zmienne `float`, `double`: 6 cyfr po przecinku, bez zbędnych zer
- wskaźniki wypisywane są jako całkowitoliczbowe adresy, w systemie szesnastkowym (z wyjątkiem typu `char`
- wskaźniki typu `char` wypisywane są jako string
- domyślne formatowanie można modyfikować przy pomocy manipulatorów

Przeładowanie operatorów << >>

- w przypadku strumienia wejściowego wszystkie wiodące znaki białe są domyslnie ignorowane
- liczba całkowita poprzedzona znakiem `0` (oczywiście bez spacji) traktowana jest jako ósemkowa, poprzedzona znakami `0b` jako dwójkowa, poprzedzona znakami `0x` jako szesnastkowa
- dla zmiennych typu `float, double` dopuszczalny jest zapis tzw. inżynierski: `-1.5e-2` (bez spacji)
- stringi odczytywane są do pierwszego znaku białego (np. spacji)
- operatory << >>, podobnie jak inne funkcje można przeładować
- Przykład `prog10.cpp`, `prog11.cpp`
- jeżeli funkcja przeładowująca jest zewnętrzna względem klasy, może oczywiście używać tylko składników publicznych. Funkcja przeładowująca może być składnikiem klasy