Operacje wejścia / wyjścia

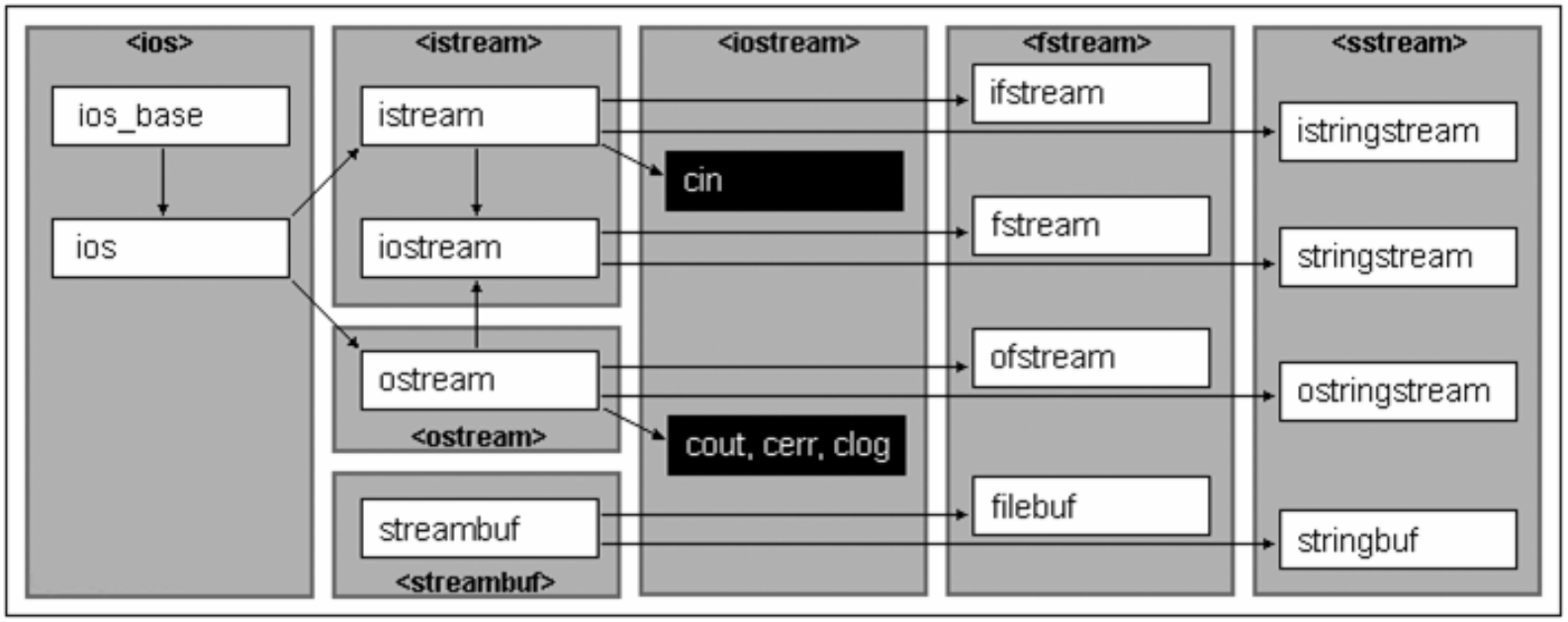
Wykład 10

Strumienie

- Wprowadzanie i wyprowadzanie informacji można potraktować jako strumień bajtów płynący od źródła do ujścia
  - Strumienie są realizowane w C++ na zasadzie klas
- Dysponujemy dwoma poziomami wczytywania i wypisywania informacji
  - Poziom niższy – elementarny, umożliwia tylko przysyłanie bajtów od źródła do ujścia
    - Nie ma interpretacji
  - Poziom wyższy – jw. z dodatkową interpretacją informacji (konwersję, sposób wyświetlania itp.)

Biblioteka do obsługi strumieni

- W C++ operacje we/wy wykonywane są za pomocą strumieni
- Hierarchia klas oraz i rozmieszczenie w plikach nagłówkowych jest następująca



Klasy niższego poziomu do obsługi strumieni

- **streambuf** - abstrakcyjna klasa bazowa, z której wywodzą się obiekty odpowiadające za bufor strumieni
- **filebuf** – instancje tej klasy odpowiadają za operacje wejścia i wyjścia dla plików oraz umożliwiają tworzenie strumieni związanych z tymi plikami
- **stringbuf** – jw. tylko w odniesieniu do tekstów, a nie plików

Standardowe wejście/wyjście

- **io_base** oraz **ios** – klasy definiujące składniki każdego strumienia, niezależnie czy to wejściowego czy wyjściowego
- **istream** – klasa udostępniająca metody do odczytu i interpretacji ze strumienia wejściowego
- **ostream** - klasa udostępniająca metody do zapisu do strumienia wyjściowego
- **iostream** - klasa udostępniająca metody do odczytu i zapisu ze strumienia wejściowego i wyjściowego, odziedziczona po klasach **istream** i **ostream**

Obiekty reprezentujące standardowe wejście i wyjście

- Standardowe kanały we/wy zdefiniowano jako obiekty globalne
  - **istream cin** – standardowe wejście (odpowiednik **stdin** z *C*), normalnie związany z klawiaturą
  - **ostream cout** – standardowe wyjście (odpowiednik **stdout** z *C*), normalnie związany z monitorem
  - **ostream cerr** – standardowe wyjście błędów (odpowiednik **stderr** z *C*), normalnie związany z monitorem. Służy do sygnalizowania błędów
    - Domyślnie nie jest buforowany
  - **ostream clog** – standardowe wyjście logowania (brak odpowiednika w *C*), normalnie związany z tym samym co **cerr**
    - Domyślnie jest buforowany
- Przykład cpp_10.1

Klasy strumieni służące do operacji na plikach i tekstach

- **ifstream** – klasa udostępniająca interfejs do odczytu danych z plików
- **ofstream** – klasa udostępniająca interfejs do zapisu danych do plików
- **fstream** – klasa udostępniająca interfejs do odczytu i zapisu danych do plików
- **istringstream** – klasa udostępniająca interfejs do manipulowania tekstami tak jakby były one strumieniem wejściowym
- **ostringstream** – klasa udostępniająca interfejs do manipulowania tekstami tak jakby były one strumieniem wyjściowym
- **stringstream** – klasa udostępniająca interfejs do manipulowania tekstami tak jakby były one strumieniem wejściowym wyjściowym

Przeładowane operatory << i >>

- Przy pracy ze strumieniami wykorzystujemy przeładowane operatory << i >> (normalnie przesunięcie bitowe w lewo i w prawo)
- Potrzebne było zdefiniowanie jakiś operatorów do wstawiania i pobierania danych ze strumieni
  - Wybór padł na << i >> ponieważ sugerują one ruch przez co ich użycie jest zgodne z naszą intuicyjną
  - Można się spotkać z angielskim nazwami tych operatorów odpowiednio *insert (put to)* i *extract (get from)*
- Zarówno << jaki i >> zwracają referencję do strumienia na którym pracują
  - Możliwe jest kaskadowe łączenie tych operatorów ze sobą

Domniemania przy pracy ze strumieniami predefiniowanymi

- Wstawianie do strumienia (wypisywanie)
  - Typy całkowite (`int, long` ...) wypisywane są w systemie dziesiętnym
  - Typy znakowe (`char`) wypisywane są jako pojedynczy znak
  - Liczby zmiennoprzecinkowe (`float, double` ...) wypisywane są z dokładnością do 6 miejsc dziesiętnych (bez zbędnych zer)
  - Wskaźniki (bez `char*`) wypisywane są szesnastkowo
  - Wskaźniki do `char` wypisywane są jako tekst, na który dany wskaźnik pokazuje
    - `static_cast<void*>(wsk); //wypisanie adresu`
- Przykład cpp_10.2

Domniemania przy pracy ze strumieniami predefiniowanymi...

- Pobieranie ze strumienia (wczytywanie)
  - Dla wszystkich wczytywanych typów wiodące białe znaki są ignorowane
    - Dotyczy także łańcuchów
  - Przy wczytywaniu typów całkowitych domyślnie oczekiwana jest liczba dziesiętna
  - Wstawianie znaków (-,+) przed liczbą możliwe jest bez odstępu (-13 ale nie - 13)
  - Wczytywanie liczby kończy się gdy napotkany zostaje znak nie będący cyfrą
    - Wyjątek 13.3e-13 (znowu bez spacji między 'e' i '-')
  - Wczytywanie tekstu kończy się po napotkaniu białego znaku
- Przykład cpp_10.3

Definiowanie własnych wersji operatorów << i >>

- W C++ możemy wykorzystać przeładowywani funkcji w stosunku do wyżej wspomnianych operatorów
  - Jedną z wad `stdio` z języka C jest brak obsługi typów zdefiniowanych przez użytkownika
- Przy przeładowywaniu tych operatorów należy pamiętać, żeby zwracały one referencję do strumienia, na którym pracują
  - `ostream& operator<<(ostream& out, const <klasa>&)`
  - `istream& operator>>(istream& in, <klasa>&)`
- Przykład cpp_10.4

Operatory wstawiania i pobierania nie mogą być wirtualne

- Brak dziedziczenia operatorów << oraz >> wynika z faktu, iż nie mogą być one funkcjami składowymi (a co dopiero wirtualnymi)
- A jednak wirtualne operatory << i >> w odniesieniu do strumieni byłyby bardzo przydatne, w szczególności jeśli mamy do czynienia z całą hierarchią klas
- Można „wirtualność" tych operatorów sobie zorganizować w stosunkowo łatwy sposób

Obsługa wyświetlania dla całej hierarchii klas

- Trzeba napisać nową wersję operatora **<<**, zaprzyjaźnioną z klasą podstawową hierarchii
  - Będzie ona wywoływać chronioną, wirtualną metodę, która z kolei dokona właściwego wyświetlenia obiektu
    - np..: `virtual void printOn(std::ostream&) const`
- W rezultacie **`operator<<`** zachowuje się tak, jakby był dynamicznie wiązany z odpowiednią klasą, pomimo tego, że jest to zwykła, zaprzyjaźniona funkcja
- Klasy potomne przeciążają metodę **`printOn`** i nie dostarczają one własnych wersji operatora **<<**
- Przykład cpp_10.5

Formatowanie

- Umożliwia nam zmianę domyślnego formatowanie w stosunku do predefiniowanych strumieni
- Sterowanie odbywa się poprzez flagi stanu formatowania zdefiniowane jako typ wyliczeniowy w klasie **`ios`**
  - **`boolalpha`** – format zapisu wartości logicznych,
  - **`skipws`** - ignoruj białe znaki,
  - **`left`** - justowanie lewe,
  - **`right`** - justowanie prawe,
  - **`internal`** - justowanie wewnętrzne,
  - **`dec`** - konwersja dziesiętna,
  - **`oct`** - konwersja ósemkowa,
  - **`hex`** - konwersja szesnastkowa,
  - **`showbase`** - pokaż podstawę konwersji,
  - **`showpoint`** - pokaż kropkę dziesiętną,
  - **`uppercase`** - wielkie litery (w liczbach),
  - **`showpos`** - znak + w liczbach dodatnich,
  - **`scientific`** - notacja wykładnicza liczb zmiennoprzecinkowych.,
  - **`fixed`** - notacja zwykła liczb zmiennoprzecinkowych.,
  - **`unitbuf`** - buforowanie,

Formatowanie…

- Modyfikacja flag za pomocą metod
  - `setf`, `unsetf` – niezbyt poręczny wymaga pamiętania nazw wspomnianych flag
    - Tego sposobu raczej nie będziemy wykorzystywać
  - Metod których nazwy są intuicyjne i wykonują zadania odpowiadające tym nazwom
    - Znacznie wygodniejszy sposób niż pierwszy
    - Posiada dodatkowe możliwości
    - `cout.width(7);`
  - Z wykorzystaniem tzw. manipulatorów – wstawiamy do strumienia specjalne kody, które zmieniają formatowanie
    - Również dobry sposób na zmianę formatowania
    - `cout << setw(7) << …;`
    - Potrzebny nagłówek `<iomanip>`

Metody służące do formatowania

- Na początku zapamiętywanie i ustawiane wzoru formatowania
  - Funkcja przeładowana **`flags`**
    - **`long flags(long wzor); //ustala flagi wg. wzoru`**
    - **`long flags(); //pobiera aktualny stan flag`**
- **`width([int,void])`** – określa minimalną liczbę znaków służącą do wypisywania liczby lub liczby znaków pobieranych ze strumienia
  - **`cout.width(7); char a[6]; cin.width(5);`**
- **`fill([char,void])`** - ustawia wypełnianie nadmiarowych miejsc na określony znak
- **`precision([int,void])`** - ustawia precyzję
- **`fill`** i **`width`** działają tylko raz i stan strumienia powraca do domniemania po wykonanej operacji
- Przykład cpp_10.6

Manipulatory bezargumentowe

- **`flush`** – powoduje natychmiastowe wypisanie bufora do strumienia wyjściowego (ma znaczenie bo domyślnie **`cout`** jest buforowany)
- **`endl`** – powoduje wypisanie znaku **`'\n'`** oraz wywołanie metody **`flush`**
- **`ends`** – powoduje wstawianie znaku **`NULL`** do strumienia (przydaje się przy wpisywaniu do tablicy)
- **`ws`** – powoduje ominięcie wszystkich białych znaków czekających na wyjęcie ze strumienia
- **`hex`** – ustawia podstawę liczb cał. na szesnastkową
- **`dec`** – ustawia podstawę liczb cał. na dziesiętną
- **`oct`** – ustawia podstawę liczb cał. na ósemkową

Manipulatory z argumentami

- `setw(int)` - robi to samo co `width`
- `setfill(char)` - robi to samo co `fill`
- `setprecision(int)` - robi to samo co `precision`
- `setiosflags()` - odpowiada `setf()`
- `resetiosflags()` - odpowiada `unsetf()`
- `setbase(int)` - odpowiada `oct, dec, hex`
- Przykład cpp_10.7

Definiowanie własnych manipulatorów

- Przydaje się w sytuacjach kiedy wielokrotnie musimy zmieniać sposób formatowania
  - ```
    ostream& scien(ostream& out)
    { out << setprcision(12)
          << setw(12)
          << resetiosflags(ios::fixed)
          << setiosflags(ios::scientific)
      return out; }
    ```
- lub po prostu chcemy ułatwić sobie pisanie
  - ```
    ostream& sep(ostream& out)
    { out << „; ”; return out; }
    ```
- Przykład cpp_10.7a

Operacje na plikach - otwieranie i zamykani strumienia

- Metoda `void open(char* name, int mode)` otwiera strumień w trybie `mode`
  - Domniemane tryby otwarcia `ios::in (ifstream)`, `ios::out (ofstream)`, dla `fstream` nie ma domniemania,
  - Wszystkie tryby (typ wyliczeniowy w `ios`):
    - `in` – plik do odczytu,
    - `out` – plik do zapisu,
    - `ate` – po otwarciu ustaw na koniec,
    - `app` – dopisywanie,
    - `trunc` – czyści plik przed zapisem,
    - `nocreate` – plik musi istnieć,
    - `noreplace` – plik nie może istnieć,
    - `binary` – tryb binarny
- `bool is_open();` - sprawdza czy plik jest otwarty
- `bool eof() const;` - sprawdza czy nie został osiągnięty koniec pliku
- `void close();` - zamyka strumień

Metody zarządzanie strumieniem wejściowym bez formatowania

- `istream& get(char& z);` - metoda wyjmuje jeden bajt ze strumienia
  - `cin.get(c);`
- `int get();`
- `istream& get(char* str, int dl, char sep='\n');`
  - W strumieniu zostaje separator
- `istream& getline(char* str, int dl, char sep='\n');`
  - Ze strumienia pobierany jest także separator
- Powyższe metody uwzględniają możliwość wystąpienia znaku końca pliku `EOF`
- Przykład cpp_10.8

Czytanie binarne i wstawianie

- `istream& read(char* str, int dl);`
  - Funkcja czyta dane bez żadnej interpretacji, nawet nie sprawdza czy wczytuje znak `EOF`, oczywiście wczyta mniej niż `dl` jeśli wystąpi `EOF`
- `istream& ignore(int dl, char sep=EOF);`
  - Ignoruje `dl` znaków
- `int gcount()`
  - Zwraca liczbę ostatnio wyjętych znaków podczas wczytywanie nieformatowanego
- `int peek()`
  - Pozwala „podglądnąć" jeden bajt czekający na wczytanie
- `istream& putback(char)`
  - Umożliwia oddanie do strumienia znaku, ale tylko jednego
- `ostream& put(char)`
  - Wstawia jeden znak
- `ostream& write(const char* str, int dl)`
  - Wysyła do strumienia znaki
- Przykład cpp_10.9

Obsługa błędów

- Flagi stanu strumienia
  - Zdefiniowane w `ios`:
    - `goodbit` – OK,
    - `eofbit` – koniec pliku,
    - `failbit` – wystąpił błąd we/wy, ale można go zresetować,
    - `badbit` – błąd uniemożliwiający dalszą pracę ze strumieniem
- Metody
  - `bool good(); bool eof(); bool fail(); bool bad();`
  - `int rdstate()` – służy do rozpoznawania wszystkich bitów naraz
  - `clear(int)` – ustawia określone lub wszystkie bity stanu
- Otwarcie nieistniejącego pliku ustawia `ios::failbit`
- Czytanie za końcem pliku ustawia `ios::eofbit`
- Podczas próby przeczytania liczby napotkamy inne znaki niż cyfry to ustawiony będzie `ios::failbit`
- Przykład cpp_10.10

Ustawianie miejsca w pliku

- Metody zwracające pozycję w pliku
  - `streampos tellg();`
    - Zwraca pozycję odczytu
  - `streampos tellp();`
    - Zwraca pozycję zapisu
  - `streampos`
    - Typ do pokazywania na położenie w pliku na ogół `long`
  - `istream& seekg(streampos, seekdir = ios::beg)`
    - Ustawia pozycję odczytu
  - `ostream& seekp(streampos, seekdir = ios::beg)`
    - Ustawia pozycję zapisu
  - `seekdir` typ wyliczeniowy zdefiniowany w `ios`:
    - `beg, cur, end`
- Przykład cpp_10.11

Powiązanie między strumieniami

- Predefiniowane strumienie **`cin`**, **`cerr`**, **`clog`** są powiązane ze strumieniem **`cout`**
  - To znaczy, że jeżeli nastąpi żądanie zapisu do **`cerr`**, **`clog`** lub odczytu z **`cin`** to najpierw zostanie wyczyszczony bufor **`cout`**
- Dowolne strumienie wyjściowe można wiązać ze sobą za pomocą funkcji
  - `ostream* tie ();`
  - `ostream* tie (ostream* tiestr);`
- Przydaje się np. w sytuacji kiedy dwa lub więcej strumieni pracują na jednym pliku
- Przykład cpp_10.11a

Formatowanie wewnętrzne

- Formatowanie wewnętrzne pozwala na pracę z tekstami tak jakby były strumieniami
  - Potrzebny jest plik nagłówkowy **<sstream>**
  - Możemy tworzyć obiekty klasy
    - **istringstream** – wejście
    - **ostringstream** – wyjście
    - **stringstream** – wejście i wyjście
  - Bardzo przydatne np. do konwersji łańcuchów znakowych do liczb itp.
- Przykład cpp_10.12

Strumienie, a wyjątki

- Po pierwsze biblioteka we/wy powstała przed standaryzacją i dlatego domyślnie nie obsługuje wyjątków
- Po drugie ze względu na to iż błędy związane ze strumieniami występują stosunkowo często może się okazać że wykorzystanie bitów stanu jest wydajniejsze
- Ale mamy możliwość zamiany bitów stanu na wyjątki
  - Należy wykorzystać metodę **`exceptions(bity)`**
  - Wyrzucane wyjątki są typu **`ios::failure`**
- Przykład cpp_10.13

Katalogi, ścieżki (i pliki)

- Biblioteka standardowa języka C++ nie umożliwia operacji na ścieżkach i katalogach
  - Główną przyczyną jest duża zależność tych operacji od systemu operacyjnego
    - Np. Struktura katalogów w UNIX i Windows są znacząco odmienne
- Korzystając z Boost Filesystem Library można operować na ścieżkach, katalogach oraz plikach
  - Należy jednak pamiętać, że nie jest to standard
  - Jednak twórcy tej biblioteki zapewniają możliwe dużą jej przenośność
- Przykład cpp_10.14