Strumienie w C++

- Strumienie
- Strumień jest obiektem kontrolującym przepływ danych
- Strumień wejściowy przenosi dane do programu
- Strumień wyjściowy wynosi dane poza program
- Strumienie dzielimy na binarne i tekstowe.
- Pliki nagłówkowe
- <iosfwd> - deklaracje zapowiadające klas strumieniowych. Jest niezbędny, bo niedozwolone jest użycie prostych deklaracji zapowiadających n.p. w postaci class ostream
- <streambuf> - definicje dla klasy bazowej bufora strumienia **basic_streambuf<>**
- <istream> - definicje dla klas obsługujących jedynie dane wejściowe **basic_istream<>** oraz dla klas obsługujących zarówno operacje wejścia, jak i wyjścia **basic_iostream<>**
- <ostream> - definicje klas strumieni wyjściowych **basic_ostream<>**
- <iostream> - deklaracje globalnych obiektów strumieni **.p. cin**, **cout**
- <iomanip> - parametryzowalne manipulatory
- <fstream> - strumienie plikowe
- <sstream>, <strstream> - strumienie wykorzystujące łańcuchy znakowe
- Klasy strumieniowe
- `streambuf` - klasa najniższego poziomu, umożliwia wykonywanie podstawowych operacji wejścia/wyjścia.
  - definiowanie własnych klas pochodnych

- klasa ta zarządza buforem
- jej funkcje składowe oferują możliwość wypełniania, opróżniania, zrzucania i innych sposobów manipulowania buforem

- Klasy strumieniowe
- `ios` - określa funkcjonowanie strumieni
  - podczas zwykłego korzystania z systemu wejścia/wyjścia języka C++ elementy klasy ios pozwalają wykonywać działania na wszystkich strumieniach;
- jest klasa bazową dla wielu innych klas pochodnych, służących do wykonywania działań na plikach dyskowych i formatowania danych w pamięci
- Klasy strumieniowe
- `istream i ostream` – dziedziczą z nadklasy `ios`
- klasy te stosuje się w celu utworzenia strumieni umożliwiających wykonywanie operacji wejścia, wyjścia
- Klasy strumieniowe
- `iostream` – dziedziczy pośrednio z nadklasy `ios` poprzez klasy `istream`, jak również `ostream`,
  - dostarcza metod wejścia - wyjścia do wypisywania danych na ekranie i odczytywania z klawiatury
- `fstream` – zapewnia wejście i wyjście
  z oraz do plików
- Klasy strumieniowe

- `iostream` – przy uruchomieniu tworzy cztery obiekty:
  - `cin`
  - `cout`
  - `cerr`
  - `clog`

- Klasa iostream
- cin - wprowadzanie danych ze standardowego wejścia
- cout - wyprowadzanie danych na standardowe wyjście
- cerr - wyjście błędów, nie buforowane, wszystko co zostanie wysłane jest wypisywane na standardowym urządzeniu błędów natychmiast
- Klasa iostream
- clog - wyjście błędów, buforowane, wprowadzanie danych na standardowe wyjście błędów, czyli ekran, dane są wprowadzone dopiero po zapełnieniu bufora
- Klasy strumieniowe
- Hierarchia klas strumieniowych realizujących operacje we/wy
- Operatory >> i <<
- Dla strumieni wejściowych pracujących w trybie tekstowym został zdefiniowany operator **>>** wyjmowania danych ze strumienia
- Dla strumieni wyjściowych pracujących w trybie tekstowym został zdefiniowany operator **<<** wstawiania danych do strumienia

- Operatory **>>** i **<<** zawsze zwracają referencję do strumieni na których pracują
- Można je łączyć kaskadowo przy czytaniu lub pisaniu
- Automatycznie dokonują konwersji z danych tekstowych na binarne i na odwrót
- Należy pamiętać o priorytecie operatorów gdy używa się wyrażeń
- Operatory strumieniowe >> i << zdefiniowane przez użytkownika
- Dla typów zdefiniowanych przez użytkownika można zdefiniować własne operatory wstawiania do i wyjmowania ze strumienia:

```
class Typ {…};
// wyjmowanie ze strumienia
istream & operator >> (istream &os, Typ &x);
//wstawianie do strumienia
ostream & operator << (ostream &os, const Typ &x);
```

- Należy pamiętać o zwróceniu strumienia na którym się pracuje
- Operatorów wstawiania i wyjmowania nie dziedziczy się
- Operatory nie mogą być wirtualne
- Operatory >> i <<
- Każdy ekstraktor musi zwracać odwołanie do obiektu typu istream. Również pierwszy parametr musi być odwołaniem do tego typu. Natomiast drugi parametr powinien być odwołaniem do zmiennej, do której mają zostać wpisane wprowadzone dane. Dzięki temu, że jest on odwołaniem, ekstraktor może go zmodyfikować.
- Ogólna postać ekstraktora wygląda następująco:

```
istream &operator>>(istream &strumień, klasa &obiekt)
{
//instrukcje specyficzne dla danej klasy
return  łańcuch; //zwrócenie odwołania do strumienia
}
```

- Operatory >> i <<
- Język C++ umożliwia bardzo proste tworzenie inserterów samodzielnie definiowanych klas.
- Ogólna postać insertera wygląda następująco:

```
ostream &operator<<(ostream &strumień, klasa &obiekt)
{
//instrukcje specyficzne dla danej klasy
return  łańcuch;  //zwrócenie odwołania do strumienia
}
```

- Właściwe zadanie wykonywane przez insertera zależy od programisty. Jedynym warunkiem stawianym przez język C++ jest konieczność zwrócenia parametru łańcuch. Całkowicie poprawne jest deklarowanie parametru obiekt jako odwołania do obiektu, a nie samego obiektu

- Sterowanie formatem
- Podczas operacji na strumieniu używane są pewne domniemania dotyczące formatu danych – domniemania te zapisane są w strumieniu we fladze stanu formatowania.
- Klasa w której umieszczono flagę stanu formatowania to `ios_base` – typ takiej flagi to `fmtflags`.
- Sterowanie formatem
- Flagi odpowiadające za sposób formatowania:
  - `skipws` - ignorowanie białych znaków;
  - justowanie `left`, `right`, `internal` (maska `adjustfield`)
  - `boolalpha` - pełne nazwy boolowskie

- `basefield` – maska, reprezentacja liczb całkowitych `dec`, `oct`, `hex`
- `showbase` - uwidocznienie podstawy reprezentacji
- `showpoint` - kropka dziesiętna
- `uppercase` - duże litery w liczbach
- `showpos` - znak + w liczbach dodatnich
- `floatfield – maska,` reprezentacja liczb rzeczywistych `scientific, fixed`
- `unibuf` - buforowanie

- Sterowanie formatem
- Zmianę reguł formatowania dokonuje się następującymi metodami:

```
fmtflags flags () const;
fmtflags flags (fmtflags fls);
fmtflags setf (fmtflags fl);
fmtflags setf (fmtflags fl, fmtflags mask);
fmtflags unsetf (fmtflags fl);
streamsize width () const;
streamsize width (streamsize w);
streamsize precision () const;
streamsize precision (streamsize p);
```

- Uwaga – metoda `width(w)` ma działanie jednorazowe.
- Sterowanie formatem
- Przykłady:

```
fmtflags f = cout.flags();
cout.unsetf(ios::boolalpha);
cout.setf(ios::showpos|ios::showpoint);
cout.setf(ios::hex,ios::basefield);
…
cout.flags(f);
```

- Manipulatory

- Manipulatory to specjalne obiekty, które można umieścić w strumieniu za pomocą operatorów >> albo <<, które powodują zmianę reguł formatowania lub inne efekty uboczne na strumieniu.
- Standardowe manipulatory bezargumentowe:
  `endl, ends,`
  `hex, dec, oct,`
  `fixed, scientific,`
  `left, right, internal,`
  `skipws, noskipws, ws,`
  `boolalpha, noboolalpha,`
  `showpoint, noshowpoint,`
  `showpos, nowhowpos,`
  `showbase, noshowbase,`
  `uppercase, nouppercase,`
  `unitbuf, nounitbuf,`
  `flush.`
- Manipulatory
- Standardowe manipulatory sparametryzowane:
  `setw(int)`
  `setprecision(int)`
  `setfill(char), setfill(wchar_t)`
  `setbase(),setbase(int)`
  `setiosflags(fmtflags)`
  `resetiosflags(fmtflags)`
- Manipulatory
- Własne manipulatory bezparametrowe definiuje się w postaci funkcji.
- Przykład:

```
ostream & tab (ostream & os)
{
    return os << ”\t”;
}
```

```
…
cout << ”x:” << tab << x << endl;
```

- Manipulatory - wybór
- Manipulatory - wybór
- Nieformatowane
  operacje we/wy
- Formatowane operacje we/wy przeprowadzane są za pośrednictwem operatorów >> i <<, które przekształcają dane z postaci tekstowej na binarną (czytanie) albo z postaci binarnej na tekstową (pisanie).
- Nieformatowane operacje we/wy są umieszczone w klasach `istream` i `ostream` (oraz uzupełnione kilkoma funkcjami składowymi w klasie `iostream`).
- Nieformatowane czytanie (wyjmowanie ze strumienia)
- Funkcje składowe wyjmujące po jednym znaku:
  `istream & get (char &);` – w przypadku końca strumienia strumień przechodzi w stan błędu
  `int get ();` – w przypadku końca strumienia funkcja zwraca wartość EOF
- Przykłady użycia:

```
char a, b, c;
cin.get(a).get(b).get(c);
…
char z;
while (cin.get(z)) {…}
…
char z;
while ((z=cin.get())!=EOF) {…}
```

- Nieformatowane czytanie

- Funkcje składowe wyjmujące wiele znaków:
  `istream & get (char *gdzie, streamsize ile, char ogr='\n');` – gdy w trakcie czytania znaków zostanie napotkany ogranicznik, to czytanie będzie przerwane (znak ogranicznika pozostanie w strumieniu)
  `istream & getline (char *gdze, streamsize ile, char ogr='\n');` – gdy w trakcie czytania znaków zostanie napotkany ogranicznik, to czytanie będzie przerwane (znak ogranicznika zostanie usunięty ze strumienia)
- Po zakończeniu czytania powyższe funkcje dopiszą na końcu danych bajt zerowy poprawnie kończący C-string (wczytanych zostanie więc maksymalnie `ile-1` znaków).
- Nieformatowane czytanie
- Funkcje zewnętrzna wyjmująca wiele znaków to:
  `istream & std::getline (isteram &we, string &wynik, char ogr='\n');` – funkcja ta nie ma limitu na liczbę wczytywanych znaków (znak ogranicznika zostanie usunięty ze strumienia).
- Przykład użycia:
```
string s;
while (getline(cin,s)) {…}
```
- Nieformatowane czytanie
- Binarne czytanie danych:
  - `istream & read (char *gdzie, streamsize ile)` – funkcja wczytuje blok znaków (gdy brakuje danych strumień przechodzi w stan błędu)
  - `streamsize readsome (char *gdzie, streamsize ile)` – funkcja wczytuje blok znaków (gdy brakuje danych strumień nie zmienia stanu)

  - `istream & ignore (streamsize ile=1, int ogr=EOF)` – funkcja pomija blok znaków

- Nieformatowane czytanie
- Binarne czytanie danych:
  - `streamsize gcount ()` – funkcja mówi, ile znaków zostało pobranych za pomocą ostatniej operacji czytania nieformatowanego
  - `int peek ()` – funkcja pozwala podejrzeć następny znak w strumieniu
  - `istream & putback (char)` – funkcja zwraca do strumienia jeden znak
  - `istream & unget ()` – funkcja zwraca do strumienia ostatnio przeczytany znak
- Nieformatowane pisanie (wstawianie do strumienia)
- Wstawianie do strumienia realizuje się za pomocą dwóch funkcji składowych:
  `ostream & put (char)` – funkcja ta wstawia do strumienia jeden znak
  `ostream & write (const char *skąd, streambuf ile)` – funkcja ta wstawia do strumienia wiele znaków
- Nieformatowane pisanie
- Przykłady użycia:

```
char napis[] = "jakiś napis";
for (int i=0; napis[i]; ++i)
    cout.put(i?' ':'-').put(napis[i]);
…
ofstream plik = …;
double e = 2.718281828459;
plik.write(reinterpret_cast<char*>(&e),sizeof(e));
```

- Operacje na plikach
- Operacje IO na plikach dla typu **char** obsługiwane są w C++ przez
  - ifstream, ofstream, fstream
- Aby obsłużyć operacje IO w pliku należy:
  - zdefiniować strumień -> określić z jakim plikiem strumień ma się komunikować
  - otworzyć plik
  - wykonać operacje wejścia-wyjścia
  - zamknąć plik
  - zlikwidować strumień
- Otwieranie i zamykanie plików
- void ifstream::open(const char* nazwa, int tryb=ios::in);
- void ofstream::open(const char* nazwa, int tryb=ios::out);
- void fstream::open(const char* nazwa, int tryb);
- Otwieranie i zamykanie plików
- ifstream, ofstream, fstream są funkcjami składowymi, które otwierają plik o podanej nazwie i łączą go ze strumieniem w trybie określonym parametrem.
- Uwaga!
  - Ponowne otwarcie pliku za pomocą **ofstream** powoduje usunięcie całej dotychczasowej zawartości pliku

- Otwieranie i zamykanie plików
- Tryb tekstowy – znaczniki są interpretowane
- Tryb binarny – wyłączona jest wszelka interpretacja znaków, nawet znaku końca pliku
- Otwieranie i zamykanie plików
- Przykłady
- ```
  std::fstream plik;
  plik.open( "nazwa_pliku.txt", std::ios::in |
  std::ios::out );
  ```
- ```
  bool good();
  bool is_open();
  std::fstream plik;
  plik.open( "nazwa_pliku.txt", std::ios::in |
  std::ios::out );
  if( plik.good() == true )
  {
      std::cout << "Uzyskano dostep do pliku!" <<
  std::endl;
      //tu operacje na pliku
  } else std::cout << "Dostep do pliku zostal
  zabroniony!" << std::endl;
  ```