Szablony

Szablony zapewniają możliwość tworzenia funkcji ogólnych (generycznych) i klas ogólnych (generycznych). W funkcjach (klasach) ogólnych typ danych, na których działa funkcja (klasa), jest określany jako parametr. Oznacza to, że możesz stworzyć funkcję (klasę) przystosowaną do pracy z kilkoma typami danych bez jawnego programowania nowej wersji funkcji (klasy) dla każdego z poszczególnych typów danych.

Funkcje ogólne

- Definiuje zestaw operacji do wykonania na różnych typach danych.

- Typ danych jest przekazywany jako parametr.

- Wiele algorytmów ma dokładnie tą samą logikę dla różnych typów danych. Na przykład, wyszukiwanie obiektu w tablice (strukturze danych), sortowanie obiektów w tablice (strukturze danych), dodawanie obiektu do tablicy (do struktury danych), usuwanie obiektu z tablicy (ze struktury danych) … .

- Użycie funkcji ogólnych (klas ogólnych) jest bardzo skuteczne, gdy algorytmy przetwarzania danych nie zależą od typów danych. Następnie kod można rozdzielić na procedury (warstwy), które nie zależą od typu danych (nazwiemy ich *algorytmami*) oraz na procedury, które odnoszą się do obsługi konkretnego typu danych.

- Część kodu, która nie zależy od typu danych, można skutecznie utworzyć w postaci funkcji ogólnych (klas ogólnych).

- Podczas tworzenia funkcji ogólnej tworzona jest funkcja, która może sama siebie przeciążać.

- Funkcji ogólne tworzą się przy użyciu kluczowego słowa *template*. Przy tym tworzony jest szkielet funkcji, w których podczas kompilacji zostaną podstawione odpowiednie typy danych - kompilator sam generuje każdą konkretną implementację dla podanego typu danych:

```
template  <class Ttype>  typ_return_value  function_name(list of arguments)
{
   //function body
}
```

- Ttype - typ danych używany przez funkcje. Ta nazwa jest używana w definicji funkcji. Ten symbol jest szablonem. Wszystkie działania algorytmu ogólnego będą wykonywane przy użyciu tego symbolu, a podczas tworzenia określonej wersji funkcji kompilator zastąpi ten symbol szablonu rzeczywistym (podstawionym) typem danych. Słowo *class* można zastąpić *typename*

```
template <typename Ttype>  typ_return_value  function_name(list of arguments)
{
   //function body
}
```

Example W21.

- Funkcja ogólna (definicję tej funkcji poprzedza słowo kluczowe *template*) nazywana jest również funkcją-szablonem. Dla każdego podstawionego typu danych kompilator tworzy określoną specjalizację tej funkcji - mówimy, że kompilator tworzy wygenerowaną funkcję. Proces generowania funkcji nazywa się *instantiation*.

- Słowo kluczowe *template* może znajdować się w innej linii:

```
template <class T>
void findmydata(T ob, size_t  len)
{
……..
}
```

- Pomiędzy linią template <class T> a linia void findmydata(T ob, size_t len) nie może pojawić się żadna instrukcja.

- W szablonie można użyć więcej niż jednego typu ogólnego

```
template <class T1, class T2>  void myfunc(T1 a, T2 b)
{
……..
}
```

- Podczas tworzenia szablonu funkcji kompilator generuje tyle różnych wersji tej funkcji, ile jest potrzebne do obsługi wszystkich różnych wywołani funkcji występujących w programie.

- Funkcji-szablony są podobne do funkcji przeciążonych, ale są bardziej ograniczone. W przypadku przeciążenia funkcji można wprowadzać dowolne zmiany w instrukcjach kodu, funkcja szablonu musi wykonywać te same instrukcje kodu dla każdej wersji danych typu ogólnego.

➢ Chociaż sama funkcja szablonu wykonuje przeciążenie w razie potrzeby, można wykonać przeciążenie jawne:

```
template <class T> void swapargs(T  &a,  T  &b)
{
      T    tmp;
      tmp = a;       // class T  must have an overload of the operator =
      a = b;
      b = tmp;
}

//Tutaj funkcja szablonu pozostanie zdefiniowana ponownie
void swapargs(double &a, double &b)
{
            double t = a;
            a = (a+b)*(a+b);
            b = t;
}

void main()
{
      double a = 10.0, b = 20.0;
      coord c1(0, 0), c2(1,1);
      swapargs(c1, c2);          //call to initial swapargs
      swapargs(a, b);             // call to overloaded swapargs
}
```

- Podany przykład nie jest typowy. Jeśli logika programu polega na tym, że każda wersja funkcji powinna wykonywać własny algorytm, bardziej naturalne jest użycie przeciążonych funkcji. Jeśli algorytm jest taki sam dla każdej wersji, a różnica dotyczy tylko typu danych, użycie szablonów jest naturalne.

- **Example W22. (Funkcja-szablon Find – zajęcia laboratoryjne)**

Ogólne klasy (klasy-szablony)

- Klasa zawiera definicje używanych przez nią algorytmów, ale typ danych, na którym operuje klasa, zostanie do niej przekazany jako parametr dopiero podczas tworzenia obiektu.

- Klasy ogólne są przydatne, gdy jeden zestaw algorytmów musi być użyty dla różnych typów danych.

- Deklaracja klasy-szablonu:

```
temlate <class Ttype> class  class_name
{
};
```

- Ttype - nazwa typu, który zostanie określony podczas tworzenia obiektu klasy.

- Tworzenie obiektu klasy-szablonu:

class_name <*typ*> objekt;

- typ - jest to typ danych, na którym ma działać klasa. Funkcje - składowe klasy-szablonu automatycznie stają się funkcjami ogólnymi, nie ma potrzeby poprzedzania ich nazwy słowem kluczowym *template* podczas deklarowania ich prototypów. Natomiast przy definicji funkcji-składowych klasy ogólnej użycie słowa kluczowego *template* jest konieczne.

Example W23.

- Klasy-szablony zapewniają możliwość tworzenia ogólnej formy algorytmu, który może być dalej używany do obsługi dowolnego typu danych. Pozbawia to programistę od pisania wielu implementacji tego samego algorytmu dla różnych typów danych.

- Klasa-szablon może mieć wiele typów danych

```
template <class T1, class T2> class myclass
{
      ……..……
};
```

- Domyślne typy danych skojarzone ze standardowymi typami danych mogą być używane w klasie-szablonie:

```
tempalte  <class T1, class T2 = int>  class   my_cl
{
……....
};

void main()
{
      my_cl< coord, double >  aa(……..);   //T1 ← coord;  T2 ← double
      my_cl <coord>   bb(……..);            //T1 ← coord;  T2 ← int
}
```

Example W24.

- Słowo kluczowe *typename* ma dwa zastosowania:
  - służy jako zamiennik słowa kluczowego *class*;
  - informuje kompilator, że nazwa użyta w deklaracji szablonu dotyczy nazwy typu, a nie nazwy obiektu.

```
template <class T> class MY_CLASS {
      typename T::Name someObj; };
```

- Name jest traktowany jako nazwa typu, a nie nazwa obiektu.

- Słowo kluczowe *export* może poprzedzać deklaracje szablonu. Daje to możliwość umieszczenia deklaracji szablonu w jednym pliku, a definicji - w innym.

- UWAGA! Kompilator programu Visual Studio nie obsługuje słowa kluczowego *export*. W przypadku programów złożonych deklaracje i definicje szablonów muszą być umieszczone w pliku *.h, a w tych plikach, w których ten szablon ma być udostępniony, należy dołączyć plik nagłówkowy. *Przy inkludowaniu takiego pliku do kilku plików źródłowych (*.cpp) nie powstaje błędu o wielokrotnej definicji tego samego obiektu, jak to by było w przypadku klasy zwyklej (nie szablonu), ponieważ jeden szablon może służyć dla tworzenia obiektów dla kilku różnych typów danych.*

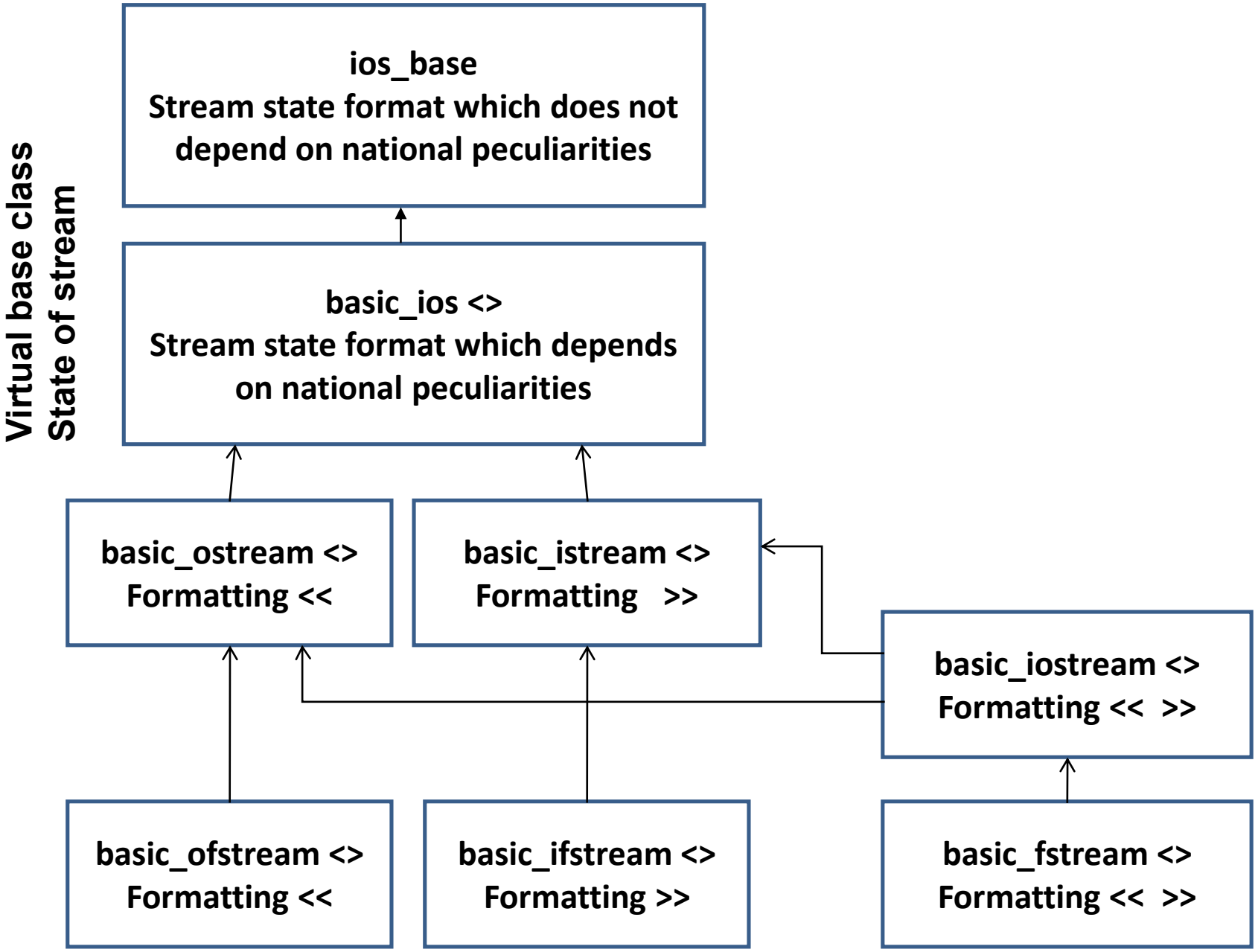
Wejście-wyjście w C ++

- C ++ obsługuje dwa kompletne systemy I/O
  - I / O języka C biblioteki CRT.
  - I / O obiektowy C++.

- Główną zaletą C++ I/O są przeciążane operatory <<, >> dla tworzonych klas. Umożliwia to wbudowanie typów danych tworzonych przez programistę w układy I/O (*strum* << my_object;).
- Strumień – to urządzenie logiczne, produkujące lub pobierające informacje.
- Wszystkie strumieni We/Wy zachowują się tak samo – system I/O daje ten samy interfejs dla różnych urządzeń fizycznych. To oznacza, na przykład, że ta sama funkcja będzie wykonywała wyprowadzenie na monitor, w plik, na drukarkę, … .
- Podczas uruchamiania programu w C++ są tworzone strumienie *cin*, *cout*, *cerr, clog* .

- Standard C++ tworzy również strumieni dodatkowe *wcin*, *wcout*, *wcerr*, *wclog* – dla *wide text format* (wchar_t) (16-bitowych) symboli.
- System I/O bazuje się na dwóch różnych hierarchiach klas-szablonów.
- Biblioteka I/O tworzy dwie oddzielne wersje hierarchii klas — jedna jest 8-bitowa, a druga 16-bitowa.

Template class	Class for 8-bit version
basic_streambuf	streambuf
basic_ios	ios
basic_istream	istream
basic_ostream	ostream
basic_iostream	iostream
basic_fstream	fstream
basic_ifstream	ifstream
basic_ofstream	ofstream


Virtual base class
State of stream
ios_base
Stream state format which does not depend on national peculiarities
basic_ios <>
Stream state format which depends on national peculiarities
basic_ostream <>
Formatting <<
basic_istream <>
Formatting >>
basic_iostream <>
Formatting << >>
basic_ofstream <>
Formatting <<
basic_ifstream <>
Formatting >>
basic_fstream <>
Formatting << >>

- Dla dostępu do poważnej klasy *ios* trzeba dołączyć nagłówek <iostream>.

Formatowanie operacji Wejścia \ Wyjścia

- Formatowanie I\O można dokonywać:
  - przy użyci składowych klasy *ios*;
  - przy wykorzystani manipulatorów – specjalnych funkcji, które można umieszczać w wyrażeniach dotyczących We\Wy.

Formatowanie przy użyci składowych *ios*

- Z każdym strumieniem związany jest zbiór flag formatowania, kontrolujących sposób formatowania informacji. To są maski bitowe.
- W klasie *ios* zadeklarowana wyliczeniowa maska bitowa fmtflags: (class ios_base, members – patrz MSDN).

- Enumeration *fmtflags*:

adjustfield	**floatfield**	**right**	**skipws**
basefild	**hex**	**scientific**	**unitbuf**
boolalpha	**internal**	**showbase**	**uppercase**
dec	**left**	**showpoint**	
fixed	**oct**	**showpos**	

flaga	ustawiona	wyzerowana
skipws	Przy wyprowadzeniu do strumienia wiodące znaki puste (spacje, tabulatory, nowej linii) są ignorowane	nie są ignorowane
left	Wyrównanie po lewej krawędzi	Jeśli wszystkie te flagi są ignorowane – ustawienia domyślne
right	Wyrównanie po prawej krawędzi	
internal	Wartości numeryczne są dopełniane tak, by zajęły całe pole – wstawiane spacje pomiędzy cyframi i znakiem liczby	

flaga	ustawiona	wyzerowana
oct	wyświetlanie wartości w systemie ósemkowym	domyślnie - syst. dziesiętnym
hex	w systemie szesnastkowym	
dec	przywrócenie do systemu dziesiętnego	----------
showbase	pokazać podstawą systemu liczenia	nie pokazywać
uppercase	w notacji naukowej wyświetlamy ‘E’, w systemie szesnastkowym – ‘X’	domyślnie: ‘e’, ‘x’
showpos	znak ‘+’ jest wyświetlany	nie wyświetlany
showpoint	wyświetla wszędzie przecinek i zero końcowe	nie wyświetla
scientific	wartości zmienno-przecinkowe są wyświetlane w notacji naukowej 3.141592e+00	kompilator sam wybiera odpowiednie notacje
fixed	wartości zmienno-przecinkowe są wyświetlane w notacji zwykłej: 3.141592	
unitbuf	bufor jest opróżniany (flush) po każdej operacji insertion (wyprowadzenie w strumień)	nie
boolalpha	wartości logiczne mogą być wprowadzone i wyprowadzone jako *false, true*	liczbami 0, 1
basefield	dec | hex | oct	nie

flaga	ustawiona	wyzerowana
adjustfield	internal | left | right	nie
floatfield	fixed | scientific	nie

ios_base class-members:

fmtflags setf(fmtflags flagi)	Zwraca poprzednie ustawienie flag i ustawia wymienione na liście argumentów. stream.setf(ios::showpos | ios::scientific); stream – strumień, do którego należy ustawienie flag, jest obiektem klasy pochodnej od klasy ios_base. *W C++ nie istnieje formatowania globalnego, każde formatowanie odbywa się dla poszczególnych strumieni.* showpos, scientific – to są wartość wyliczeniowa w klasie ios, dla tego operator zakresu :: jest konieczny, inaczej kompilator „nie widzi” tych wartości.
void unsetf(fmtflags flagi)	Flagi, wymienione na liście *flagi*, są wyzerowane. Wszystkie pozostałe flagi - bez zmian.

fmtflags flags();	Zwraca bieżący stan flag, nic nie zmienia w ustawieni flag ios::fmtflags f = cout.flags();

fmtflags flags(fmtflags *f*);	Ustawia wszystkie flagi, zdefiniowane w szablonie fmtflags *f,* zwraca poprzednie ustawienie flag
	
	ios::fmtflags f = ios::showpos | ios::showbase;
	cout.flags(f);

streamsize width(streamsize w);	domyślnie przy wyprowadzeni wartość zajmuje tyle pozycji, ile ma symbolów. Funkcje width() zadaje minimalną szerokość pola w, zwraca poprzednią szerokość pola. *Po wykonaniu każdego wyprowadzenia szerokość pola pozostaje ustawiona jako wartość domyślna.* Reszta pola (nie zajęta symbolami) będzie wypełniona symbolem zapełnienia (domyślnie – spacja). Jeśli ilość symbolów przekracza ustawioną szerokość pola, będzie wyprowadzone tyle symbolów, ile ma wartość.
streamsize precision(streamsize p);	Domyślnie liczby zmiennoprzecinkowe są wyprowadzone z dokładnością 6 cyfr. Dokładność można zmienić przy pomocy funkcji precision(). Zwraca poprzednią dokładność.

Funkcja fill() jest składową klasy pochodnej basic_ios:

template <class Elem, class Traits> class basic_ios : public ios_base

char fill(char ch);	Domyślnie wypełnienie pustych pozycji odbywa się spacjami. Funkcja fill() nadaje możliwość wypełnić te pozycji symbolem ch

Example W25. (samodzielnie)

Manipulatory We \ Wy

Manipulatory – to funkcje specjalne, które można włączać do wyrażeń We \ Wy.
Dla dostępu do manipulatorów z parametrami trzeba dodać nagłówek <iomanip>.

Standard manipulators:

manipulator	przeznaczenie	We \ Wy
boolalpha	Włącza flagę boolalpha	We \ Wy
dec	Włącza flagę dec	We \ Wy
endl	Wyprowadza znak nowej linii i opróżnia strumień	Wy
ends	Wyprowadza znak '\0'	Wy
fixed	Włącza flagę fixed	Wy
flush	Opróżnia strumień	Wy

manipulator	przeznaczenie	We \ Wy
hex	Włącza flagę hex	We \ Wy
internal	Włącza flagę internal	Wy
left	Włącza flagę left	Wy
noboolalpha	Wyłącza flagę boolalpha	Wy
noshowbase	Wyłącza flagę showbase	Wy
noshowpoint	Wyłącza flagę showpoint	Wy
noskipws	Wyłącza flagę skipws	Wy
nounitbuf	Wyłącza flagę unitbuf	Wy
nouppercase	Wyłącza flagę uppercase	Wy
oct	Włącza flagę oct	We \ Wy
resetiosflags(fmtflags f)	Wyłącza flagi wymienione w f	Wy
right	Włącza flagę right	Wy
scientific	Włącza flagę scientific	Wy
setbase(int base)	Przypisuje podstawie systemu liczenia wartość, wskazaną w base	We \ Wy
setfill(int ch)	Definiuje jako znak wypelnienia ch	Wy
setiosflags(fmtflags f)	Włącza flagi wymienione w f	We \ Wy

manipulator	przeznaczenie	We \ Wy
setprecision(int p)	Ustala wartość dokładności	Wy
setw(int w)	Definiuje szerokość pola jako w	Wy
showbase	Włącza flagę showbase	Wy
showpoint	Włącza flagę showpoint	Wy
showpos	Włącza flagę showpos	Wy
skipws	Włącza flagę skipws	Wy
unitbuf	Włącza flagę unitbuf	Wy
uppercase	Włącza flagę uppercase	Wy
ws	Opuszcza puste znaki wiodące	We

Example:

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

int main()
{
    cout << hex << 100  << endl;
    cout << setfill('?') << setw(10) << 2343.0;
}
output:  64
         ??????2343
```

- Manipulatory występują w wyrażeniach złożonych i często okazują się bardziej przydatne dla zastosowania, niż składowe klasy ios.

- Jeśli manipulator nie pobiera argumenty (na przykład, endl, hex), nie występują nawiasy. Dzieje się tak dla tego, że jest to adres funkcji przekazywany do przeciążonego operatora << .

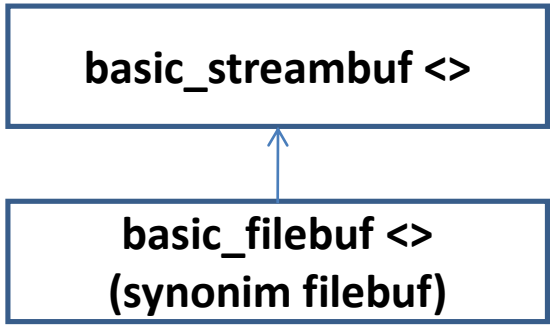
- Manipulatory We \ Wy wpływają tylko na strumień, w wyrażenie dla którego są włączone. Na inne strumienie, otwarte w programie, nie wpływają.

Example W26. (Samodzielnie)

Operacje Wejścia / Wyjścia, przeprowadzane na plikach

- Pliki dyskowe mają wielu unikalnych cech, dla tego w układzie We\Wy C++ są tworzone klasy specjalne dla pracy z plikami: *ofstream*, *ifstream*, *fstream* (slide 12) – hierarchie klas. Nagłówek <fstream> .

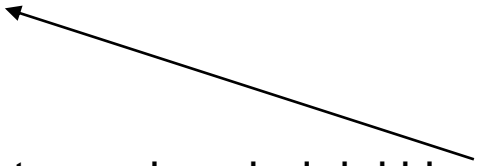
- Dla zarządzania niskopoziomowego strumieniami dla plików istnieje klasa szablonowa basic_filebuf



Opening and closing a file.

- Otwieranie i zamykanie pliku. Otwieranie pliku polega na połączeniu go ze strumieniem. Są trzy rodzaje strumieni:

  - wejściowe (obsługiwany przez klasę *ifstream*)
  - wyjściowy (*ofstream*)
  - wejście\wyjście (*fstream*)



- Otwieranie pliku powstaje przy wywołaniu konstruktora odpowiedniej klasy lub przy jawnym użyciu metody klasy open(…).

Example W27.

- Metody open:

void basic_ifstream::open(const char *_Filename, ios_base::openmode _Mode = ios_base::in, int _Prot = (int)ios_base::_Openprot);

void basic_ofstream::open(const char *_Filename, ios_base::openmode _Mode = ios_base::out, int _Prot = (int)ios_base::_Openprot);

void basic_fstream::open(const char *_Filename, ios_base::openmode _Mode = ios_base::in | ios_base::out, int _Prot = (int)ios_base::_Openprot);

Opening and closing a file.

- Tryb otwierania pliku zależy od wartości ios_base openmode _Mode.

```
class ios_base {
public: typedef implementation-defined-bitmask-type openmode;
static const openmode in; static const openmode out; static const openmode ate; static const
openmode app; static const openmode trunc; static const openmode binary; // ... };
```

ios_base::app	Przed każdym zapisem w plik wskaźnik pozycji pliku będzie przestawiony na koniec pliku (tryb dodawania w koniec pliku)
ios_base::ate	Przy otwierani pliku wskaźnik pozycji będzie przestawiony na koniec pliku, przecież zapis \ odczyt może być dokonywany w dowolnym miejscu pliku
ios_base::binary	Tryb binarny. Domyślnie pliki są otwarte w trybie tekstowym. Plik, otwarty w trybie tekstowym, „widzi” znaki formatowania i znaki specjalne. Plik, otwarty w trybie binarnym, „nie widzi” formatowania.
ios_base::in	Plik jest otwarty dla odczytu (extraction)
ios_base::out	Plik jest otwarty dla zapisu (insertion)
ios_base::trunc	Zawartość pliku istniejącego pozostaje zniszczona, plik w trakcie otwierania będzie ucięty do zerowej długości

basic_filebuf::open

ios_base::in	becomes "r" (open existing file for reading).
ios_base::out **ios_base::out | ios_base::trunc**	becomes "w" (truncate existing file or create for writing).
ios_base::out | ios_base::app	becomes "a" (open existing file for appending all writes)
ios_base::in | ios_base::out	becomes "r+" (open existing file for reading and writing).
ios_base::in | ios_base::out | ios_base::trunc	becomes "w+" (truncate existing file or create for reading and writing).
ios_base::in | ios_base::out | ios_base::app	becomes "a+" (open existing file for reading and for appending all writes).

_Prot - parametr domyślnej ochrony otwierania plików, równoważna parametrowi *shflag* w *_fsopen*(…..).

- **Sprawdzenie poprawności otwarcia pliku.**

  - Używamy konstruktor klasy strumienia. W przypadku niepowodzenia strumień jest ustawiony na *false*:

```
fstream iof(“myfile.dat”, ios_base::in | ios_base::out);
if(! iof)
{
        //plik nie pozostał otwarty
}
//OK
```

  - Używamy metodę bool basic_filebuf::is_open(): zwraca *true*, jeśli plik pozostał otwarty, *false* – niepowodzenie.

```
fstream iof(“myfile.dat”, ios_base::in | ios_base::out);
if(!iof.is_open())
{
        //plik nie pozostał otwarty
}
//OK.
```

- **Zamykanie plików:** void basic_fstream :: close( );

Niesformatowane i binarne operacje wejścia-wyjścia

- W funkcjach klasycznych We\Wy C\C++ są zrealizowane operacje na bajtach. Dla tego że *sizeof*(*char*) = 1 B, w prototypach tych funkcji jest używany typ *char*.

- Będziemy otwierali pliki w trybie binarnym (ios_base::binary).

- Funkcje *put*, *get* z biblioteki Run Time C ++ dla plików binarnych wykonują zapis / odczyt jednego bajtu informacji:

istream & get (char & ch);
ostream & put (char ch);

Example W28.

- Funkcje *read*, *write* z biblioteki Run Time C ++ służą dla zapisu/odczytu bloków bajtów:

```
basic_istream  & read( char_type *_Str,   streamsize _Count );
basic_ostream & write( const char_type *_Str,   streamsize _Count );
```

_Str jest wskaźnikiem do bufora pamięci rzutowanego na char * lub const char *, a _Count podaje liczbę bajtów do zapisu / odczytu.

Jeśli przy odczycie koniec pliku występuje wcześniej niż będzie przeczytane _Count bajtów, odczyt pozostaje przerwany, a w buforze _Str znajduje się tyle bajtów, ile faktycznie pozostało przeczytane. Dla tego żeby wyjaśnić, ile bajtów pozostało przeczytane, istnieje funkcja

```
streamsize basic_istream :: gcount( ) const;
```

- Dla wyładowania buforu We\Wy w plik, używamy

```
basic_ostream & basic_ostream::flush ();
```

Nie stosujemy flashowania pliku niepotrzebnie - znacznie spowalnia to działanie programu. Podczas zamykania pliku dane wyładowują się z bufora We\Wy i są zapisywane do pliku.

- Dostęp bezpośredni. W C++ plik ma 2 wskaźnika pozycji – jeden określa pozycje pliku, gdzie nastąpi kolejna operacje wejścia, a drugi – wyjścia. Dla przesunięcia wskaźników pozycji plików mamy 2 układy funkcji: xxxg() (get - odczyt), xxxp() (put – zapis).

- Przesuwa wskaźnik pozycji pliku o wartość _Off odnośnie _Way

```
basic_istream & basic_istream::seekg  ( off_type _Off, ios_base::seekdir _Way );
basic_ostream& basic_ostream:: seekp ( off_type _Off, ios_base::seekdir _Way );
```

  - _Way == ios_base::beg – odnośnie początku pliku,
  - _Way == ios_base::end – odnośnie końca pliku,
  - _Way == ios_base::cur – odnośnie bieżącej pozycji.

- Ustawia wskaźnik pozycji pliku w pozycje, podaną przez _Pos:

```
basic_istream & basic_istream::seekg  ( pos_type _Pos );
basic_ostream& basic_ostream:: seekp( pos_type _Pos );
```

- Wskazuje, w której pozycji znajduje się wskaźnik pozycji pliku:

```
pos_type  basic_istream  :: tellg( );
pos_type  basic_ostream :: tellp( );
```

Example:

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

int main ( )
{
   using namespace std;
   ifstream file;
   char c, c1;

   file.open( "basic_istream_seekg.txt" );
   if(!file) return;
   file.seekg(2);   // chars to skip
   file >> c;
   cout << c << endl;

   file.seekg( 0, ios_base::beg );
   file >> c;
   cout << c << endl;

   file.seekg( -1, ios_base::end );
   file >> c1;
   cout << c1 << endl;
   file.close();
}
```

Text file:
basic_istream_seekg.txt:

0123456789

Output:

2
0
9

- Sprawdzenie stanu wejścia \ wyjścia. Bieżący stan systemu We \ Wy jest przechowywany w obiekcie typu *ios_base* :: *iostate*:

```
class ios_base {
public:
        typedef implementation-defined-bitmask-type iostate;
        static const iostate badbit;
        static const iostate eofbit;
        static const iostate failbit;
        static const iostate goodbit;
        //. . .
};
```

Jest to maska bitowa opisująca obiekt, który może przechowywać informacje o stanie strumienia. Odrębne wartości flag bitowych to:

badbit, utrata integralności bufora strumienia.
eofbit, jest odczytany markier końca pliku.
failbit, niepowodzenie wyodrębnienia prawidłowego pola. Na przykład, failbit = 1 jeśli przy wprowadzeniu *int* okazał się symbol, który nie może być konwertowany poprawnie na typ *int* (Przykład IO_1).

Przydatną jest wartość **goodbit**, gdy żaden z wcześniej wymienionych bitów nie jest ustawiony. W przeciwnym przypadku goodbit ma gwarantowane zero.

- Funkcja *ios_base*::*iostate basic_ios*::*rdstate*() *const*; zwraca obiekt *iostate* zawierający te flagi bitowe.

- Inny sposób sprawdzenia:

```
bool  basic_ios::bad( ) const;      //true, jeśli flaga badbit jest ustawiona
bool  basic_ios::eof( )  const;     //true, jeśli flaga eofbit jest ustawiona
bool  basic_ios::fail( )   const;    // true, jeśli flaga failbit jest ustawiona
bool  basic_ios::good( ) const;    // true, jeśli flaga goodbit jest ustawiona
```

- wyzerowanie flag błędów:

```
void  basic_ios::clear( iostate _State = goodbit );
```

Jeśli jako argument jest podana flaga *goodbit*, wszystkie flagi będą wyzerowane. Jeśli jako argument podać inną flagę, tylko ta flaga będzie wyzerowana.

Example IO_1.