342

C++ - strumienie

Operatory **<<** i **>>**

Aby wysłać dane do strumienia należy użyć operatora **<<**

rezultatem działania przeciążonego operatora jest odniesienie do tego samego obiektu/strumienia. Kolejność realizacji działań jest następująca (*nawiasy są zbędne – tutaj umieszczone tylko dla ujawnienia kolejności*):

((cout << x) << y) << z;

Aby odczytać dane ze strumienia należy użyć operatora **>>**

rezultatem działania również tego przeciążonego operatora jest odniesienie do tego samego obiektu/strumienia:

((cin >> x) >> y) >> z;

343

C++ - strumienie

Dla każdego wbudowanego typu danych istnieje przeciążony operator

`operator>>` i **`operator<<`**

Można też przeciążać ten operator na własne potrzeby.

344

C++ - strumienie

Przykład – przeciążanie operatora << :

```
class Integer {
      int i;
public:
      Integer(int v): i(v) {};
   friend ostream& operator<< (ostream & b , const
  Integer & p);
};

ostream& operator << (ostream & b , const Integer & p) {
        return b << p.i;
}
```


345

C++ - strumienie

W przypadku przeciążania należy zachować następujące wymagania:

1. pierwszym parametrem musi być niestała referencja do strumienia (`istream` dla wejścia lub `ostream` dla wyjścia); strumień ten nie może być `const`, gdyż przetwarzanie danych strumienia powoduje zmianę jego stanu
2. wykonane musi być wstawienie bądź pobranie danych do lub ze strumienia
3. zwrócić trzeba referencję do strumienia, tak aby można było łączyć sekwencje operacji na strumieniu w jedną instrukcję

346

C++ - strumienie

Domyślne formatowanie wartości:

przy wypisywaniu:

- wartości całościowe (`int`, `short`, itd.) w systemie dziesiętnym
- wartości znakowe jako pojedyncze znaki
- wartości zmiennoprzecinkowe (`float`, `double`, itd.) w systemie dziesiętnym z precyzją 6 cyfr znaczących (np.. 200.0/3 daje 66.6667, 1.129995 zostanie najpierw zaokrąglone do 1.13000 a potem wypisane jako 1.13, ale 1.129994 – jako 1.12999)
- wartości wskaźnikowe (adresy) jako dodatnie liczby całkowite reprezentujące adres w systemie szesnastkowym z prefiksem '`0x`'
- wartości wskaźnikowe typu '`char*`' jako C-napisy
- wartości logiczne '`bool`' jako 0 lub 1

347

C++ - strumienie

Domyślne formatowanie wartości:

przy wczytywaniu:

- wiodące białe znaki są pomijane
- każda następna niepusta sekwencja białych znaków jest interpretowana jako koniec danych. Znaki te pozostają w buforze i będą pominięte, jako wiodące przy następnym czytaniu
- liczby całkowite są wczytywane w postaci dziesiętnej aż do napotkania odstępu lub znaku nie będącego cyfrą – ten znak jest pozostawiany w buforze i będzie pierwszym znakiem wczytanym przy następnej operacji wczytania (napis 128-25 zostanie wczytany jako dwie liczby: 128 i -25)
- liczby zmiennoprzecinkowe są wczytywane w formacie liczb całkowitych bez kropki, w formacie z kropką dziesiętną i w notacji „naukowej", tj. 1e-1 lub 1.20e+2, co oznacza odpowiednio: 0.1 lub 120.1)
- wartości logiczne '`bool`' mają postać literałów 0 i 1

C++ - strumienie

Obsługa błędów strumieni

Klasa `ios_base` zawiera cztery flagi, których można użyć do badania stanu strumienia:

- `badbit` – pojawił się jakiś poważny błąd (być może związany ze sprzętem)
- `eofbit` – pojawił się koniec danych wejściowych (albo fizyczny koniec pliku odpowiadającego strumieniowi, albo użytkownik zakończył strumień konsoli, np. znakiem Ctrl-Z albo Ctrl-D)
- `failbit` – wystąpił błąd operacji we/wy, najprawdopodobniej wskutek nieprawidłowych danych (np. podczas odczytu liczby pojawiły się litery). Strumień nadal może być używany. Flaga ta ustawiana jest także w przypadku końca danych wejściowych
- `goodbit` – wszystko w porządku, nie wystąpiły żadne błędy. Jeszcze nie pojawił się sygnał końca danych wejściowych

C++ - strumienie

ustawienia tych bitów sprawdzamy za pomocą odpowiednich metod:

- `if (s.bad())`
  stan strumienia nieustalony, nie należy go więcej używać (ustawiony `badbit`)
- `if (s.eof())`
  rozpoznano koniec pliku (ustawiony `eofbit`)
- `if (s.fail())`
  ostatnia z wykonanych operacji nie powiodła się (ustawiony albo `failbit` albo `badbit`)
- `if (s.good())`
  wszystko OK.

Aby wyzerować flagi należy to zrobić ręcznie wywołując:

```
s.clear();
```


C++ - strumienie

Zwykle nie interesuje nas sprawdzenie w kodzie programu poszczególnych flag ale po prostu sprawdzenie, czy wszystko jest w porządku. Wtedy w wyrażeniu logicznym wywoływany jest operator konwersji ze wskaźnika na strumień do wartości logicznej

```
ios_base::operator void*()
```

Np. w wyrażeniu:

```
if (s << x) …
```

`s << x` zwraca referencję do `s`, która jest niejawnie konwertowana do `void*` za pomocą

```
ios_base::operator void*()
```


C++ - strumienie

Metoda `operator void*()` wywołuje na rzecz strumienia metodę `good()` i zwraca jej wynik – wskaźnik `void*`, który przyjmuje NULL, kiedy wystąpił jakikolwiek błąd (metoda `fail` zwróciła `true`) albo inną wartość (nie-NULL) kiedy jest OK.

```
int i;
while (myStream >> i)
  cout << i << endl;
```


C++ - strumienie

W C++11 operator konwersji ze wskaźnika na strumień do wartości logicznej

```
ios_base::operator void*()
```

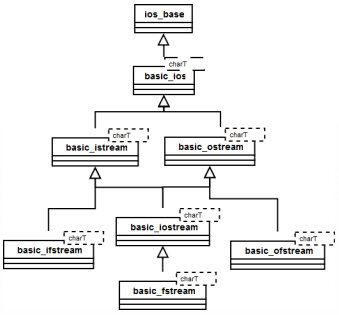
został zastąpiony operatorem:

```
explicit operator bool() const;
```

Skutek działania – taki sam.

C++ - strumienie

Strumienie związane z plikami
- obiekty fstream, ifstream i ofstream:



354

C++ - strumienie

Przetwarzanie plików za pomocą biblioteki 'iostream' jest znacznie łatwiejsze i bezpieczniejsze niż korzystanie z biblioteki 'stdio' w C:

- w celu otwarcia pliku wystarczy stworzyć obiekt – cała prace wykona konstruktor
- nie trzeba jawnie zamykać pliku – zajmie się tym destruktor

Obiekty zdefiniowane są na podstawie szablonów tak jak poniżej:

```
typedef basic_ifstream<char> ifstream;
```

Deklaracje w kodzie programu:

```
ifstream in("Test1.txt");
ofstream ou("Test2.txt", ios::app);
```


355

C++ - strumienie

Tryby otwarcia pliku:

ios::in – plik jest otwierany jako strumień do odczytu. *Tego trybu używa się do strumieni ofstream, aby uniknąć nadpisania już istniejącego pliku*

ios::app – otwiera plik do zapisu w trybie dołączania

ios::ate – otwiera istniejący plik (wejściowy lub wyjściowy) i przechodzi na jego koniec

ios::trunc – jeśli plik istnieje, usuwa jego dotychczasową zawartość

ios::binary – otwiera plik w trybie binarnym, trybem domyślnym jest tryb tekstowy

ios::out – otwiera plik do zapisu. Jeśli flaga użyta zostanie do strumienia `ofstream` bez flagi `ios::app`, `ios::ate` lub `ios::in`, zakłada się użycie `ios:trunc`

Flagi można łączyć operatorem bitowej alternatywy: **|** np.: **out|app**

356

C++ - strumienie

ios_base Flag combination					stdio equivalent
binary	in	out	trunc	app	
		+			"w"
		+		+	"a"
		+	+		"w"
	+				"r"
	+	+			"r+"
	+	+	+		"w+"
+		+			"wb"
+		+		+	"ab"
+		+	+		"wb"
+	+				"rb"
+	+	+			"r+b"
+	+	+	+		"w+b"

Jeżeli tryb otwarcia jest inny niż którakolwiek z kombinacji wymienionych powyżej, otwarcie pliku kończy się niepowodzeniem

357

STRUMIENIE

Formatowanie strumieni wyjściowych

358

C++ - strumienie

Formatowanie strumieni wyjściowych

Każda klasa reprezentująca strumień ma tzw. ***flagi***, opisujące stan strumienia.
W szczególności sposób działania operatorów we/wy określony jest aktualnym stanem flag stanu formatowania.

359

C++ - strumienie

```
#include <iostream>
fmtflags flags( ) const;
```

metoda zwraca dotychczasowy stan ustawień

Przykład:

```
ios_base::fmtflags flagi = cout.flags();
cout << flagi << endl;
```


C++ - strumienie

```
#include <iostream>
fmtflags flags( fmtflags _Fmtfl );
```

metoda modyfikuje stan ustawień na nowy i zwraca stan sprzed modyfikacji

Przykład:

```
ios_base::fmtflags flagi = cout.flags();
flagi |= ios::hex;
cout.flags( flagi );
```

Co to za stała `ios::hex` *?*

C++ - strumienie

Formatowanie strumieni wyjściowych

Możliwe wartości flag formatowania zostały zdefiniowane w klasie bazowej '`ios`' jako składowe statyczne, dlatego odwołujemy się do nich poprzez operator zakresu klasy, np.: `ios::left, ios::scientific`, itp.

Te składowe statyczne są, jak i cała flaga stanu formatowania, typu '`long`'.

Reprezentacja bitowa poszczególnych składowych statycznych zawiera jeden (z reguły jeden) ustawiony bit (jedynkę) a pozostałe nieustawione.

Składowe statyczne można łączyć za pomocą operatora alternatywy bitowej – 'lub'.

C++ - strumienie

Definicja standardowa C++ (Solaris, Microsoft C++ - dawniej)

```
enum {
        skipws          = 0x0001, // ignoruj białe znaki
        left            = 0x0002, // justuj do lewej
        right           = 0x0004, // justuj do prawej
        internal        = 0x0008, // justuj do środka
        dec             = 0x0010, // wypisuj dziesiętnie
        oct             = 0x0020, // wypisuj ósemkowo
        hex             = 0x0040, // wypisuj szesnastkowo
        showbase        = 0x0080, // pokazuj podstawę
        showpoint       = 0x0100, // pokazuj kropkę dziesiętną
        uppercase       = 0x0200, // wypisuj duże litery w liczbach
        showpos         = 0x0400, // wypisuj znak + w liczbach dodatnich
        scientific      = 0x0800, // notacja naukowa (wykładnicza)
        fixed           = 0x1000, // notacja zwykła
        unitbuf         = 0x2000, // nie buforuj strumienia
        stdio           = 0x4000  // ustaw tryb kompatybilności z stdio
    };
```


C++ - strumienie

Wersja Visual Studio 2010:

```
#define _IOSskipws      0x0001   // ignoruj białe znaki
#define _IOSunitbuf     0x0002   // nie buforuj strumienia
#define _IOSuppercase   0x0004   // wypisuj duże litery w liczbach
#define _IOSshowbase    0x0008   // pokazuj podstawę
#define _IOSshowpoint   0x0010   // pokazuj kropkę dziesiętną
#define _IOSshowpos     0x0020   // wypisuj znak + w liczbach dodatnich
#define _IOSleft 0x0040  // justuj do lewej
#define _IOSright       0x0080   // justuj do prawej
#define _IOSinternal    0x0100   // justuj do środka
#define _IOSdec 0x0200   // wypisuj dziesiętnie
#define _IOSoct 0x0400   // wypisuj ósemkowo
#define _IOShex 0x0800   // wypisuj szesnastkowo
#define _IOSscientific  0x1000   // notacja naukowa (wykładnicza)
#define _IOSfixed       0x2000   // notacja zwykła
#define _IOShexfloat    0x3000   // added with TR1/
#define _IOSboolalpha   0x4000   // wartości bool jako napisy true/false
#define _IOS_Stdio      0x8000   // ustaw tryb kompatybilności z stdio
```


C++ - strumienie

```
static const _Fmtflags skipws = (_Fmtflags)_IOSskipws;
static const _Fmtflags unitbuf = (_Fmtflags)_IOSunitbuf;
static const _Fmtflags uppercase = (_Fmtflags)_IOSuppercase;
static const _Fmtflags showbase = (_Fmtflags)_IOSshowbase;
static const _Fmtflags showpoint = (_Fmtflags)_IOSshowpoint;
static const _Fmtflags showpos = (_Fmtflags)_IOSshowpos;
static const _Fmtflags left = (_Fmtflags)_IOSleft;
static const _Fmtflags right = (_Fmtflags)_IOSright;
static const _Fmtflags internal = (_Fmtflags)_IOSinternal;
static const _Fmtflags dec = (_Fmtflags)_IOSdec;
static const _Fmtflags oct = (_Fmtflags)_IOSoct;
static const _Fmtflags hex = (_Fmtflags)_IOShex;
static const _Fmtflags scientific = (_Fmtflags)_IOSscientific;
static const _Fmtflags fixed = (_Fmtflags)_IOSfixed;
static const _Fmtflags hexfloat = (_Fmtflags)_IOShexfloat;  // added with TR1/
static const _Fmtflags boolalpha = (_Fmtflags)_IOSboolalpha;
static const _Fmtflags _Stdio = (_Fmtflags)_IOS_Stdio;
static const _Fmtflags adjustfield = (_Fmtflags)(_IOSleft | _IOSright | _IOSinternal);
static const _Fmtflags basefield = (_Fmtflags)(_IOSdec | _IOSoct | _IOShex);
static const _Fmtflags floatfield = (_Fmtflags)(_IOSscientific | _IOSfixed);
```

C++ - strumienie

```
ios_base::fmtflags flagi
       = cout.flags();
cout << flagi << endl;


W oknie konsoli:
513
```

```
#define _IOSskipws      0x0001    // ignoruj białe znaki
#define _IOSunitbuf     0x0002    // nie buforuj strumienia
#define _IOSuppercase   0x0004    // wypisuj duże litery w liczbach
#define _IOSshowbase    0x0008    // pokazuj podstawę
#define _IOSshowpoint   0x0010    // pokazuj kropkę dziesiętną
#define _IOSshowpos     0x0020    // wypisuj znak + w liczbach dodatnich
#define _IOSleft        0x0040    // justuj do lewej
#define _IOSright       0x0080    // justuj do prawej
#define _IOSinternal    0x0100    // justuj do środka
#define _IOSdec         0x0200    // wypisuj dziesiętnie
#define _IOSoct         0x0400    // wypisuj ósemkowo
#define _IOShex         0x0800    // wypisuj szesnastkowo
#define _IOSscientific  0x1000    // notacja naukowa (wykładnicza)
#define _IOSfixed       0x2000    // notacja zwykła
#define _IOShexfloat    0x3000    // added with TR1/
#define _IOSboolalpha   0x4000    // wartości bool jako napisy true/false
#define _IOS_Stdio      0x8000    // ustaw tryb kompatybilności z stdio
```


366

C++ - strumienie

```
ios_base::fmtflags flagi
       = cout.flags();
cout << flagi << endl;
flagi ^= ios::dec;
flagi |= ios::hex;
cout.flags( flagi );
cout << flagi << endl;

W oknie konsoli:
513
801

(2049 dec = 801 hex)
```

```
#define _IOSskipws      0x0001    // ignoruj białe znaki
#define _IOSunitbuf     0x0002    // nie buforuj strumienia
#define _IOSuppercase   0x0004    // wypisuj duże litery w liczbach
#define _IOSshowbase    0x0008    // pokazuj podstawę
#define _IOSshowpoint   0x0010    // pokazuj kropkę dziesiętną
#define _IOSshowpos     0x0020    // wypisuj znak + w liczbach dodatnich
#define _IOSleft        0x0040    // justuj do lewej
#define _IOSright       0x0080    // justuj do prawej
#define _IOSinternal    0x0100    // justuj do środka
#define _IOSdec         0x0200    // wypisuj dziesiętnie
#define _IOSoct         0x0400    // wypisuj ósemkowo
#define _IOShex         0x0800    // wypisuj szesnastkowo
#define _IOSscientific  0x1000    // notacja naukowa (wykładnicza)
#define _IOSfixed       0x2000    // notacja zwykła
#define _IOShexfloat    0x3000    // added with TR1/
#define _IOSboolalpha   0x4000    // wartości bool jako napisy true/false
#define _IOS_Stdio      0x8000    // ustaw tryb kompatybilności z stdio
```


367

C++ - strumienie

Dostępne flagi formatowania:

ios::skipws – zignoruj przy wczytywaniu wiodące białe znaki (*domyślnie*: TAK)

ios::dec, ios::hex, ios::oct – podstawa dla wczytywanych/pisanych liczb całkowitych: dziesiętna (*domyślnie*), szesnastkowa lub ósemkowa. Razem tworzą pole podstawy **ios::basefield**.

Najwyżej jedna z nich może być ustawiona, jeżeli żadna nie jest, to domyślnie odczyt jest dziesiętny.

368

C++ - strumienie

ios::scientific, ios::fixed – format dla wypisywanych liczb zmiennoprzecinkowych. Razem tworzą pole formatu **ios::floatfield**. Najwyżej jedna z nich może być ustawiona, jeżeli żadna nie jest, to domyślnie użyty będzie format ogólny, który implementacji pozostawia sposób zapisu w zależności od wartości liczby tak, żeby zapis z ustalona precyzją był najkrótszy.

```
ios_base::fmtflags flagi = cout.flags();
double x = 2000+1.0/3.0;
cout<< "domyslnie: " << x << endl;
flagi |= ios::scientific;
cout.flags( flagi );
cout << "scientific: " << x << endl;
flagi ^= ios::scientific;
flagi |= ios::fixed;
cout.flags( flagi );
cout<< "fixed: " << x << endl;
```

```
domyslnie: 2.00033e+003
scientific: 2.00033e+003
fixed: 2000.33333
```


369

C++ - strumienie

ios::boolalpha – określa, czy wartości logiczne wypisywać jako 0/1 (domyślnie), czy słowami false/true

ios::showbase – przy wypisywaniu zawsze pokaż podstawę (wiodące zero lub 0x w systemie ósemkowym i szesnastkowym)

ios::showpoint – zawsze wypisz kropkę dziesiętną i końcowe zera w części ułamkowej (domyślnie: NIE)

ios::showpos – pisz znak '+' przed liczbami dodatnimi (domyślnie: NIE)

ios::uppercase – litery 'e' w zapisie naukowym i 'x' w zapisie szesnastkowym pisz jako duże 'E' i 'X' (domyślnie: NIE)

ios::unitbuf – opróżnij bufor po każdej operacji zapisu

370

C++ - strumienie

Dostępne flagi formatowania:

ios::left, ios::right, ios::internal – wyrównanie przy przypisaniu do lewej, prawej albo obustronnie, czyli „znak do lewej, liczba do prawej". Na przykład, jeżeli zapisujemy liczbę -123 w polu o szerokości 8 znaków, to stosując te trzy sposoby wyrównania otrzymalibyśmy

```
|-123     |
|    -123|
|-    123|
```

Te trzy flagi razem tworzą pole justowania **ios::adjustfield**. Najwyżej jedna z nich może być ustawiona. Jeżeli żadna nie została ustawiona to domyślnie wyrównanie jest do prawej.

Aby ustawienie odniosło skutek, trzeba jeszcze ustawić szerokość pola dla wypisywanej wartości (*o tym będzie na następnych slajdach*)

371

C++ - strumienie

Ustawianie i gaszenie flag

Ustawianie i gaszenie flag przez działania na zmiennej pomocniczej za pomocą operatorów bitowych jest męczące i gmatwa kod.

Zamiast tego, na rzecz obiektu reprezentującego strumień można wywołać metody, które ustawią lub zgaszą odpowiednią flagę:

void setf(fmtflags _*Mask* **);**

fmtflags setf(fmtflags _*Mask*, **fmtflags** _*Unset* **);**

C++ - strumienie

Operatory logiczne na zmiennej `flag`

```
ios_base::fmtflags flagi;
flagi = cout.flags();
double x = 2000+1.0/3.0;
cout<< "domyslnie: " << x << endl;

flagi |= ios::scientific;
cout.flags( flagi );

cout << "scientific: " << x << endl;

flagi ^= ios::scientific;
flagi |= ios::fixed;
cout.flags( flagi );

cout<< "fixed: " << x << endl;
```

Metoda `setf`

```
double x = 2000+1.0/3.0;
cout<< "domyslnie: " << x << endl;

cout.setf(ios::scientific);

cout << "scientific: " << x << endl;

cout.setf(ios::fixed, ios::scientific);

cout<< "fixed: " << x << endl;
```


C++ - strumienie

Gaszenie flag

Aby tylko zgasić flagę, dana jest metoda `unsetf`.

void unsetf(fmtflags _*Mask* **);**

Przykład:

```
double x = 2000+1.0/3.0;
cout<< "domyslnie: " << x << endl;
cout.setf(ios::scientific);
cout << "scientific: " << x << endl;
cout.unsetf(ios::scientific);
cout.setf(ios::fixed);
cout<< "fixed: " << x << endl;
```


C++ - strumienie

Kopiowanie flag:

basic_ios& copyfmt(const basic_ios& _*Right* **);**

Kopiuje ustawienia flag z jednego strumienia do drugiego

Przykład:

```
ofstream ofx( "test.txt" );          // deklarujemy nowy strumień
ofx.setf(ios::hex, ios::basefield);  // gasimy cały basefield
                    // i ustawiamy tryb szesnastkowy (trick!)
int j = 10;
cout << j << endl;
cout.copyfmt( ofx );
cout << j << endl;
```

```
Wyjście:    10
            a
```


C++ - strumienie

Ustawianie szerokości, wypełnienia i dokładności:

streamsize width( ) const;

streamsize width(streamsize _*Wide* **);**

Metoda zwraca minimalną szerokość pola dla wartości lub ustawia jego szerokość. Domyślną wartością szerokości pola wydruku jest zero, czyli każda wypisywana liczba zajmuje tyle znaków, ile jest potrzebne, ale nie więcej:

```
cout.width( 20 );
ios_base::fmtflags starawart = cout.setf(ios::showbase);
cout.setf(ios::hex, ios::basefield);
int x = 19;
cout << x << endl;
cout << x << endl;
cout.flags(starawart);
cout << cout.width( ) << endl;
```

```
Wyjście:        0x13
                0x13
                0
```


C++ - strumienie

Ustawianie szerokości, wypełnienia i dokładności:

streamsize width( ) const;

streamsize width(streamsize _*Wide* **);**

```
char napis[10];
cin.width(sizeof(napis)); // max. szerokość dla wejścia
cin >> napis;
cout << napis << endl;
```

```
Okno konsoli: qwertyuiopasdfghjkl
              qwertyuio
```

Daje możliwość zabezpieczenia przed wpisaniem przez użytkownika większej liczby znaków niż maksymalny rozmiar bufora (w przykładzie powyżej zaakceptował tylko 9 znaków)

C++ - strumienie

Ustawianie szerokości, wypełnienia i dokładności:

streamsize precision( ) const;

streamsize precision(streamsize *_Prec* **);**

Metoda zwraca precyzję liczb zmiennoprzecinkowych lub ją ustawia.
Precyzja określa, ile cyfr zostanie wykorzystanych przy wypisywaniu liczby (łącznie, tj. przed i po przecinku):

```
float i = 313.12345;
cout.precision( 4 );
cout << i << endl;
```

```
Wyjście: 313.1
```


378

C++ - strumienie

Ustawianie szerokości, wypełnienia i dokładności:

- **char_type fill( ) const;**
- **char_type fill(char_type** *_Char* **);**

Metoda zwraca znak wypełniający lub go ustawia

```
float i = 313.12345;
cout.width( 20 );
cout.fill('*');
cout.precision( 4 );
cout << i << endl << i << endl;
```

```
Wyjście:      **************313.1
              313.1
```


379

C++ - strumienie

Dane wejściowe pobierane wierszami:

int_type get( );

basic_istream& get(char_type& *_Ch* **);**

basic_istream& get(char_type *_Str*, **streamsize** *_Count*);

basic_istream& get(char_type *_Str*, **streamsize** *_Count*,
char_type *_Delim* **);**

Przerywa swoje działanie zaraz po znalezieniu ogranicznika, ale tego ogranicznika już nie pobiera ze strumienia.

```
char c[10];
c[0] = cin.get( ); // pobiera jeden element i zwraca pobraną wartość
cin.get( c[1] );   // pobiera jeden element i wpisuje do argumentu
cin.get( &c[2], 3);// pobiera do napotkania znaku '\n', ale max. 2
cin.get( &c[4], 4, '7');// pobiera do napotkania znaku '7', ale max. 3
cout << c << endl;
```


380

C++ - strumienie

Dane wejściowe pobierane wierszami:

basic_istream& getline(char_type *_Str*, **streamsize** *_Count* **);**

basic_istream& getline(char_type *_Str*, **streamsize** *_Count*,
char_type *_Delim* **);**

Przerywa swoje działanie zaraz po znalezieniu ogranicznika, którego pobiera ze strumienia.

```
int main( )
{
   char c[10];

    // pobiera max. 4 znaki do napotkania '2' (piąty to '\0' - zamyka C-napis)
   cin.getline( &c[0], 5, '2' );

   cout << c << endl;
}
```


381

C++ - strumienie

Dane wejściowe pobierane wierszami:

Funkcja globalna **`getline()`** z pliku nagłówkowego **`<string>`**
wprowadza dane ze strumienia do obiektu typu **`string`**. Wczytuje do napotkania ogranicznika, znaku końca pliku, lub gdy liczba wczytanych znaków przekroczy **`is.max_str`**:

```
int main()
{
   string s1;
   cout << "wprowadź tekst (uzyj <spacja> jako znaku końca): ";
   getline(cin,s1, ' ');
   cout << "Wprowadziłeś: " << s1 << endl;;
}
```


382

C++ - strumienie

```
const int SZ = 100; // rozmiar bufora
char buf[SZ];

ifstream in("Test1.txt");
ofstream ou("Test2.txt", ios::app);

while(in.getline(buf, SZ)) {
        // uwaga: getline usuwa \n
   char* cp = buf;
   while(*cp != ':')
        ++cp;
   *cp = 0;
   cp += 2;
   cout << cp << endl;
   ou << cp << ":" << buf << endl;
}
```

Test1.txt:

Iksiński:00
Kowalski:01
Nowak:02

Okno terminala:

0
1
2

Co zawiera Test2.txt?

383

C++ - strumienie

Inne metody zdefiniowane w klasie **ios**:

basic_istream& ignore(streamsize *_Count* **= 1,**
int_type *_Delim* **= traits_type::eof( ));**

pomija określoną liczbę znaków w strumieniu wejściowym, lub nieokreśloną – aż do wystąpienia określonego znaku:

```
char charDynamicznyStos[10];
cout << "Napisz 'abcdef': ";
cin.ignore( 5, 'c' );
cin >> charDynamicznyStos;
cout << charDynamicznyStos;

Wyjście: def
```


C++ - strumienie

Inne metody zdefiniowane w klasie ios:

int_type peek( );

zwraca jeden znak ze strumienia (w postaci zmiennej typu `int`, aby mógł to być również znak EOF). Znak pozostaje w strumieniu i będzie pierwszym znakiem wczytanym przy następnej operacji czytania

```
char c[10], c2;
cout << "Napisz 'abcde': ";
c2 = cin.peek( );
cin.getline( &c[0], 9 );
cout << c2 << " " << c << endl;

Wyjście: Napisz 'abcde': abcde
a abcde
```


C++ - strumienie

Inne metody zdefiniowane w klasie ios:

basic_istream& putback(char_type *_Ch* **);**

wstawia znak *_Ch* do strumienia; będzie on pierwszym znakiem wczytanym przez następną operację czytania. Zwraca odnośnik do strumienia na rzecz którego została wywołana. Nie do każdego strumienia i nie zawsze można wstawić znak

```
char c[10], c2, c3;
c2 = cin.get( );   // użytkownik nacisnął klawisz 'q'
c3 = cin.get( );   // użytkownik nacisnął klawisz 'w'
cin.putback( c2 );
cin.getline( &c[0], 9 );
cout << c << endl;

Wyjście:    qw
            q
```


C++ - strumienie

Inne metody zdefiniowane w klasie ios:

basic_istream& unget( );

wstawia ostatnio przeczytany znak z powrotem do strumienia; będzie on pierwszym znakiem wczytanym przez następną operację czytania. Zwraca odnośnik do strumienia na rzecz którego został wywołany.

```
char c[10], c2;

cout << "Napisz 'abc': ";
c2 = cin.get( );
cin.unget( );
cin.getline( &c[0], 9 );
cout << c << endl;

Wyjście:    Napisz 'abc': abc
            abc
```


C++ - strumienie

Przykład:

```
cout << "Wpisuj linie tekstu. Zakończ znakiem końca pliku\n" <<
        "(Ctr-Z w Windows, Ctr-D w Linuksie). Komentarze\n" <<
        "od \'//\' do końca linii będą pomijane\n\n";
char c;
while ( cin.get(c) )
{
  if ( c == '/' )
    if ( cin.peek() == '/')   // sprawdź, czy następny to też slash
    {
      cin.ignore(1024,'\n');  // pomiń 1024 znaki lub do końca linii
      cout << endl;
      continue;
    }
  cout << c;
}
```


C++ - strumienie

Działania na flagach stanu:

Zerowanie flagi błędu:

void clear(iostate ***_State*****=goodbit);**

zeruje flagi błędu (wywołanie bez argumentu) i ewentualnie ustawia niektóre z nich podane w argumencie wywołania.

C++ - strumienie

Działania na flagach stanu:

iostate rdstate( ) const;

odczytuje stan bitów dla flag

```
void TestFlags( ios& x )
{
   cout << ( x.rdstate() &
        ios::badbit ) << ' ';
   cout << ( x.rdstate() &
        ios::failbit ) << ' ';
   cout << ( x.rdstate() &
        ios::eofbit ) << endl;
   cout << endl;
}
```

```
int main( )
{
   ofstream ofx( "test.txt" );
   ofx.clear( );
   TestFlags( ofx );
   ofx.clear(  ios::badbit |
               ios::failbit |
               ios::eofbit );
   TestFlags( ofx );
```

```
Wyjście:    0 0 0
            1 4 2
```


390

C++ - strumienie

Inne metody zdefiniowane w klasie ios:

basic_ostream<Elem, Traits> *tie( ) const;

basic_ostream<Elem, Traits> *tie(basic_ostream<E, T> *_Str);

gwarantuje kolejność wykonywania operacji na strumieniach

Pierwsza metoda zwraca wskaźnik wiązania dla strumienia, druga umieszcza wskaźnik do strumienia w swoim polu wskaźnika wiązania. W ten sposób gwarantuje, że działania na powiązanym drugim strumieniu nie rozpoczną się, zanim w pełni nie zostaną zakończone działania na pierwszym.

```
int i;
cin.tie( &cout );
cout << "Wprowadź liczbę:";
cin >> i;
```

Przykład jest sztuczny, ponieważ domyślnie strumienie cin i cout sq powiązane

391

C++ - strumienie

Inne metody zdefiniowane w klasie ios:

Metoda

```
operator void*()              // (C++98)
explicit operator bool()      // (C++11)
```

wywołuje na rzecz strumienia metodę **good()**:

```
int i;
while (myStream >> i)
  cout << i << endl
```


392

C++ - strumienie

Inne metody zdefiniowane w klasie ios:

Metoda

```
bool operator!( ) const;
```

wywołuje na rzecz strumienia metodę **fail()**:

```
if (!myStream) {
      // wykonaj obsługę tej sytuacji
      …
}
```


393