Programowanie obiektowe

Paweł Mikołajczak, 2019

6. Operacje wejścia/wyjścia
Part I

Web site: informatyka.umcs.lublin.pl

Wykład 6 *Operacje wejścia/wyjścia*

- ***Wstęp.***
- ***Stara i nowa biblioteka wejścia/wyjścia.***
- ***Operacje wejścia/wyjścia języka C.***
- ***Operacje wejścia/wyjścia języka C++.***
- ***Klasy strumieni***
- ***Plikowe operacje wejścia/wyjścia języka C++.***
- ***Globalne obiekty strumieni.***
- ***Obsługa wyjścia – obiekt cout, flagi formatowania.***
- ***Obsługa wyjścia – manipulatory do formatowania.***
- ***Obsługa wejścia – obiekt cin, flagi formatowania.***
- ***Obsługa wejścia – obiekt cin, stan strumienia.***
- ***Obsługa wejścia – obiekt cin, łańcuchy w stylu C.***

System wejścia /wyjścia

Język C++ nie zawiera instrukcji służących do obsługi operacji wejścia i wyjścia. Odpowiednie funkcje i makrodefinicje realizujące te zadania zostały umieszczone w bibliotekach standardowych. Język C++ jest nadzbiorem języka C, co powoduje, że programista ma do czynienia z dwoma praktycznie różnymi systemami wejścia/wyjścia.

Te dwa systemy możemy sklasyfikować jako:

- proceduralny system wejścia/wyjścia
- obiektowy system wejścia/wyjścia

Pierwszy zestaw został przejęty z języka C. W języku C operacje wejścia/wyjścia obsługują funkcje biblioteczne. Najczęściej używane to funkcje typu **printf(), scanf(), gets, puts**, itp. Ponieważ wymienione funkcje znajdują się w plikach nagłówkowych, konieczne jest ich dołączanie do programów. Najczęściej mamy do czynienia z plikami takimi jak **<stdio.h>** oraz **<conio.h**>.

System wejścia /wyjścia

Ponieważ obsługa wejścia/wyjścia opracowana dla języka C nie była doskonała, twórcy języka C++ dodatkowo opracowali drugi, ***nowy,*** zorientowany obiektowo system wejścia/wyjścia. W tym systemie programista otrzymuje zestaw klas, które są zorganizowane w skomplikowany, hierarchiczny sposób, z wykorzystaniem dziedziczenia. Przy pomocy tych klas możemy realizować najróżniejsze operacje wejścia/wyjścia. Metody tych klas i funkcje zaprzyjaźnione zapewniają bezpieczną obsługę tzw. ***strumieni.*** Strumienie są reprezentowane przez odpowiednie obiekty należące do klas obiektowej biblioteki wejścia/wyjścia. Strumień wejściowy reprezentowany jest przez predefiniowany obiekt **cin**, a strumień wyjściowy przez **cout**. Operacje wejścia/wyjścia korzystają najczęściej z dwóch przeciążonych operatorów **<<** i **>>**. Do programów korzystających z obiektowego systemu wejścia/wyjścia należy włączyć odpowiednie pliki nagłówkowe, najpopularniejszym jest plik **iostream.h** (tak zwana stara biblioteka C++), obecnie rekomendujemy nową bibliotekę C++ z plikiem **iostream** (bez .h)

System wejścia /wyjścia

Standardowa biblioteka C++ zawiera klasy dla obsługi tekstowych strumieni wejścia/wyjścia.

Zanim opracowano aktualny standard ANSI/ISO, większość kompilatorów języka C++ była dostarczana z biblioteką klas powszechnie znaną jako biblioteka ***iostream.*** Aby nie było nieporozumień, starszą bibliotekę nazywa się tradycyjna biblioteką C++ iostream (ang. *as traditional iostreams*) w odróżnieniu do aktualnej wersji (nowa wersja), którą nazywamy standardową biblioteką C++ iostream (ang. *standard iostreams*).

Symbolami biblioteki C++ iostream są obiekty **cout** i **cin** z operatorami **<<** oraz **>>**.

Stara i nowa biblioteka wejścia/wyjścia

W kolejnych latach nastąpił dalszy rozwój języka C++, opracowano nową wersję biblioteki wejścia/wyjścia. W rezultacie programista ma do dyspozycji dwie biblioteki:

- ***starą bibliotekę systemu wejścia/wyjścia C++***
- ***nową bibliotekę systemu wejścia/wyjścia C++***

Z punktu widzenia programisty, obie biblioteki obiektowe różnią się niewiele. Jak to zwykle jest z językiem C/C++, nowa biblioteka jest nadzbiorem starej biblioteki. Praktyczne różnice są widoczne w nazwie pliku nagłówkowego. Stara wersja biblioteki związana jest z plikiem **<iostream.h>,** nową bibliotekę obsługuje plik **<iostream>.** Merytorycznie, nowa biblioteka wejścia/wyjścia C++ zawiera nowe funkcje i wprowadza nowe typy danych. Bardzo istotne jest to, że w nowej bibliotece wszystkie symbole są zdefiniowane w przestrzeni nazw **std**. Jak wiemy, symbole i konstrukcje występujące w programie mogą być grupowane za pomocą przestrzeni nazw. Pozwala to na unikniecie konfliktów nazw (gdy wprowadzamy do programu różne biblioteki i inne pliki) ale także tworzyć logiczne powiązania pomiędzy różnymi symbolami.

Operacje wejścia/wyjścia

Kultowy program produkujący napis „**Hello, World**” przedstawimy w kilku postaciach ilustrujących zagadnienia związane z systemami wejścia/wyjścia.

W kolejnych przykładach pokazano związane z operacjami wejścia/wyjścia podstawowe pliki nagłówkowe, funkcje, metody, obiekty i operatory oraz użycie przestrzeni nazw **std** w nowej bibliotece C++.

Wydruk a) strukturalne we/wy, język C

```
#include <stdio.h>
main()
{    printf("Hello, World");
}
```

Wydruk b) obiektowe we/wy, stary styl, język C++

```
#include <iostream.h>
main()
{  cout << "Hello, World";
}
```

Operacje wejścia/wyjścia

Wydruk c) obiektowe we/wy, nowy styl, język C++

```
#include <iostream>
main()
{ std::cout << "Hello, World";
}
```

Wydruk d) obiektowe we/wy, nowy styl, język C++

```
#include <iostream>
using namespace std;
main()
{  cout << "Hello, World";
}
```

Wydruk e) obiektowe we/wy, nowy styl, język C++

```
#include <iostream>
using std::cout;
main()
{  cout << "Hello, World";
}
```

Operacje wejścia/wyjścia

Wydruk f) mieszane we/wy, nowy styl, język C++ i C

```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{ cout << " C++ Hello, World";
   printf("\n C Hello, World");
   getche( );
}
```

Prezentowany tutaj styl raczej nie jest polecamy !

Operacje wejścia/wyjścia języka C

W ciągu najbliższych kilku (kilkunastu lat) języki C i C++ będą współistnieć. Może się zdarzyć, że programista będzie musiał modyfikować programy napisane w języku C, stąd znajomość systemu wejścia/wyjścia języka C ***jest niezbędna***.

Obecnie (2017 rok) oprogramowanie w systemach wbudowanych w zasadzie pisane jest wyłącznie w języku C!

Oprócz tych technicznych argumentów, należy pamiętać, że istnieje niepisana zasada, że każdy programista stosujący język C++ musi znać język C. W kolejnych krótkich przykładach przypomnimy zasady obsługi wejścia/wyjścia języka C.

W języku C, do obsługi znaków służą między innymi funkcje **getch(), getche(), getchar(),** i **putchar().** Funkcja **getchar()** oraz **getche()** wczytuje znak z klawiatury, funkcja **putchar()** wypisuje znak na ekranie. Popularna funkcja **printf()** służy do wyświetlania napisów i wyników na ekranie.

Operacje wejścia/wyjścia języka C

W większości kompilatorów prototypy tych funkcji znajdują się w plikach nagłówkowych **stdio.h** i **conio.h**. W niektórych kompilatorach nazwy tych funkcji mają z przodu znak podkreślenia. W kompilatorze Microsoft Visual C++ mamy nazwy takie jak **_getch()** oraz **_getche().** Programy pisane w języku C++ mogą korzystać z plików nagłówkowych nowego stylu, w takim przypadku stara nazwa pliku poprzedzona jest literą **c**, na przykład mamy plik **<cstdio.h>**. W wielu kompilatorach możemy wykorzystać pliki bez rozszerzenia **h**, tzn. umieszczać pliki w postaci **<cstdio>.** Często włączenie tylko pliku **<iostream>** pozwala na korzystanie z usług klasycznych funkcji wejścia/wyjścia języka C, takich jak na przykład **printf().** Funkcje **printf()** oraz **scanf()** wykonują operacje formatowanego wejścia/wyjścia. Oznacza to, że możemy wprowadzać dane różnych typów, należy tylko przekazać tym funkcjom odpowiednie specyfikatory formatu.

Należy jednak zawsze upewniać się jak jest zorganizowana biblioteka wejscia/wyjscia.

Operacje wejścia/wyjścia języka C

```
// Pobieranie i wyświetlanie znaków, język C

#include <stdio.h>
#include <conio.h>

int main()
{
  int zn;
  printf("\n napisz imie, Enter konczy\n ");
  do
   { zn = getchar();
     putchar(zn);
   } while (zn != '\n');
  getche();            // ”przytrzymanie” ekranu
  return 0;
}
```

Operacje wejścia/wyjścia języka C

Napisy wprowadzane z klawiatury możemy wczytywać za pomocą funkcji **gets().** Wpisywanie znaków kończy naciśniecie klawisza ***Enter***. Jeżeli nastąpi pomyłka podczas wpisywania znaków, można skorygować błąd korzystając z klawisza ***Backspace.***

Po wprowadzeniu łańcucha do tablicy znaków, nasz program sprawdza, jaką literą są zapisane znaki. Gdy znak pisany jest małą literą, następuje zamiana na dużą literę. Na ekranie otrzymamy wprowadzony łańcuch pisany dużymi literami.

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
int main()
{  char *zn, s[80];
   zn = s;
   puts ("napisz tekst, Enter konczy ");
   gets(s);
   while(*zn)
      { if (islower( *zn)) *zn = toupper(*zn);
        putchar(*(zn++));
      }
   getche();
   return 0;
}
```

Operacje wejścia/wyjścia języka C

Na końcu łańcucha zwracanego przez funkcję **gets()** umieszczany jest znak końca łańcucha **'\0'**. Prototyp funkcji **gets()** ma postać:

```
char *gets ( char *str ) ;
```

gdzie **str** to wskaźnik do tablicy znaków, w której należy umieścić łańcuch wpisany z klawiatury. Makrodefinicja **islower(zn)** służy do testowania czy otrzymany znak jest małą literą, jej prototyp znajduje się w pliku **<ctype.h>.** Makrodefinicja zwraca wartość niezerową, gdy **zn** jest małą literą. Funkcja **toupper(zn)** zwraca dużą literę będącą odpowiednikiem **zn** pod warunkiem, że **zn** jest literą. W przeciwnym przypadku **zn** jest zwracane w niezmienionej postaci. Prototyp tej funkcji znajduje się w pliku **<ctype.h>.**

W języku C/C++ łańcuch jest obsługiwany jako tablica znaków. We fragmencie naszego programu:

```
char *zn, s[80];
 zn = s;
 puts ("napisz tekst, Enter konczy ");
 gets(s);
```

zadeklarowano tablicę znaków **s[80]** oraz zmienną wskaźnikową **zn**. Zmienna wskaźnikowa **zn** otrzymuje adres pierwszego elementu tablicy **s**. Wiemy, że pomiędzy tablicami i wskaźnikami istnieje ścisły związek, dzięki czemu do elementów tablicy możemy się odwoływać przy pomocy dwóch metod – korzystając z indeksów oraz wskaźników. Odwoływanie się do elementów tablicy jest bardziej czytelne i zrozumiałe, z kolei użycie wskaźników jest o wiele szybszą metodą. W wielu aplikacjach szybkość wykonywania programu spełnia kluczową rolę, dostęp do elementów tablicy realizowany jest najczęściej przy pomocy wskaźników.

Operacje wejścia/wyjścia języka C

W wielu programach potrzebne są funkcje wykonujące rozmaite manipulacje na łańcuchach. W bibliotece języka C mamy wiele takich funkcji, duża ich ilość jest umieszczona w pliku nagłówkowym **<string.h>.** Kolejny program ilustruje wykorzystanie funkcji obsługującej łańcuchy – funkcję **strlen().**

```
//Pobieranie łańcucha i wyświetlanie znaków, język C
#include <stdio.h>
#include <conio.h>
#include <string.h>

int main()
{
  char *s = "Ala i sok";
  printf("%s\n", s);
  for (int i = strlen(s) - 1;  i > -1;  i--)
      printf("%c", s[i]);
  getche();
  return 0;
}
```

Operacje wejścia/wyjścia języka C

W instrukcji:

```
char *s = "Ala i sok";
```

mamy przykład deklaracji łańcucha z jednoczesną inicjalizacją. Tworzona jest tablica znaków **s[ ],** do jej elementów możemy odwoływać się poprzez indeksy.

W instrukcjach:

```
for (int i=strlen(s)-1; i>-1; i--)
    printf("%c", s[i]);
```

przetwarzamy nasz napis w ten sposób, że na ekranie ukaże się on pisany wspak. Funkcja **strlen()** wylicza długość łańcucha, nie licząc jego znaku końca. Do poszczególnych znaków odwołujemy się przy pomocy indeksów.

Po uruchomieniu programu mamy następujący wynik:

```
Ala i kos
Sok i alA
```

Operacje wejścia/wyjścia języka C

W języku C jedną z najczęściej używanych funkcji formatowanego wyjścia jest funkcja **printf().** Jest to niesłychanie rozbudowana funkcja o olbrzymiej ilości parametrów. W kolejnym przykładzie omówimy pewne elementy funkcji **printf(),** pozwalające ustalać format drukowanych danych. W naszym programie mamy zadeklarowana strukturę o nazwie ***personel***, w której przechowujemy dane osobowe. Program korzysta z funkcji **drukuj_dane(),** przy jej pomocy wyświetlane są dane osobowe na ekranie. Do wyświetlania danych wybrano funkcję **printf().** Prototyp tej funkcji umieszczony jest w pliku nagłówkowym **<stdio.h>.**

Operacje wejścia/wyjścia języka C

```
// Wyświetlanie danych struktury, funkcja printf(), język C
#include <stdio.h>
#include <conio.h>

struct personel
{ char imie_P[15], imie_D[15], nazwisko[20];
  int rok_Ur;
  double pensja;
};
void drukuj_dane(struct personel *ws)
{ printf("\n%-20s %-15s %-15s %6d %8.2lf PLN",
          ws->nazwisko,
          ws->imie_P,
          ws->imie_D,
          ws->rok_Ur,
          ws->pensja);
}
int main()
{
  struct personel p1 = {"Jan", "Maria", "Kowalski", 1947, 3500.50};
  drukuj_dane(&p1);
  getche();
  return 0;
 }
```

Operacje wejścia/wyjścia języka C

Funkcja **drukuj_dane()** wysyła do urządzenia wyjściowego (na ekran) informacje umieszczone w strukturze ***personel*** wskazanej parametrem tej funkcji:

```
void drukuj_dane ( struct personel  *ws )
{
  //.....................................
}
```

Bezpośrednio wydrukiem danych zajmuje się funkcja **printf(),** w której łańcuch formatujący ma dość skomplikowaną postać:

```
printf("\n%-20s %-15s %-15s %6d %8.2lf PLN", .....................
```

sekwencja znaków formatujących

```
%-20s
```

oznacza, że będzie wydrukowana wartość zmiennej (znak %), zmienna jest typu tablica znaków, czyli łańcuch (znak s), na zapisanie łańcucha zarezerwowano pole o długości 20 znaków w linijce tekstu (znak 20), łańcuch justowany jest lewostronnie.

Operacje wejścia/wyjścia języka C

Napis **%-15s** różni się od poprzedniego tylko tym, że na zapisanie łańcucha zarezerwowano pole o szerokości 15 znaków. Sekwencja znaków formatujących

%6d

oznacza, że będzie wyświetlona wartość liczby całkowitej (znaki % i d), na zapis liczby zarezerwowano pole o szerokości 6 znaków (są to cyfry), justowanie jest prawostronne. Sekwencji znaków

%8.2lf

oznacza zlecenie wyświetlenia wartości zmiennej typu **double** (znaki % oraz lf), na wyświetlenie liczby zarezerwowano pole o szerokości 8 znaków (wliczając w to kropkę dziesiętną i jedno pole przeznaczone na znak). Po kropce dziesiętnej wyświetlone mają być dwie cyfry.

Po uruchomieniu tego programu otrzymamy następujący wynik:

Kowalski Jan Maria 1947 3500.50 PLN

Operacje wejścia/wyjścia języka C

Należy pamiętać, że ogólna postać sekwencji formatujących jest bardzo skomplikowana i przedstawia się następująco:

%[flagi][szerokość][.precyzja][F][N][h][l][L]znak_typu

- % - sekwencje formatujące rozpoczynają się tym znakiem
- flagi – dodatkowe możliwości (np. justowanie)
- szerokość – długość zarezerwowanego pola o podanej szerokości do pomieszczenia wartości
- precyzja – oznacza ilość cyfr po kropce dziesiętnej
- F, N, h, l, L, modyfikatory typu argumentu (np. wskaźnik, long int, long double, itp.)
- znak_typu – określa typ argumentów.

Operacje wejścia/wyjścia języka C

- Jak już mówiliśmy, w programach pisanych w języku C++ możemy używać konstrukcji biblioteki standardowej We/Wy języka C.
- Wielu programistów ciągle preferuje funkcje wejścia/wyjścia zawarte w bibliotekach języka C. Pisanie programów strukturalnych przy pomocy języka C++ uzasadnia takie podejście, najczęściej wykorzystywane są funkcje takie jak **printf(), scanf()** czy **gets()**. Ponieważ główne funkcje tej biblioteki umieszczone są w pliku **<stdio.h>** , mówimy, że programiści pracują z biblioteką C stdio, w odróżnieniu do programistów preferujących pracę z biblioteką C++ iostream. Oprócz zalet biblioteka C stdio ma też wiele braków. Najczęściej wskazuje się na brak stabilności, jeżeli chodzi o typy oraz na trudności obsługi obiektów, gdy stosujemy własne klasy.
- Rozważmy wywołanie klasycznej funkcji z biblioteki stdio z użyciem standardowej biblioteki iostream.

Operacje wejścia/wyjścia języka C

```
//Wyświetlanie danych, funkcja fprintf(), język C
#include <stdio.h>
int main()
{
  int d = 1;
  char m[15] = "Maj";
  fprintf(stdout, "%d %s", d, m);
  getchar();
  return 0;
}
```

Po uruchomieniu programu mamy prawidłowy wydruk:

1 Maj

Operacje wejścia/wyjścia języka C

- Możemy zapytać, co się stanie, gdy przypadkowo zmienimy kolejność argumentów w funkcji **fprinf(),** to znaczy zamiast ” **d, m**” napiszemy ”**m, d**”? Wszystko może się zdarzyć, od dziwnego wydruku po załamanie się systemu. Taka sytuacja nigdy nie wystąpi, gdy używamy standardowej biblioteki iostream:

  cout << d << ” ” << m << endl ;

- Ponieważ mamy do czynienia ze zbiorem przeciążonych operatorów przesunięcia – **operator<< (),** właściwy operator zawsze będzie wywołany. Funkcja **cout << d** wywoła **operator<< (int),** a funkcja **cout << m** wywoła **operator<< (char*).**

  W ten sposób kolejność danych nie gra roli, standardowa biblioteka ***iostream*** zapewnia bezpieczeństwo ze względu na typ.

- Kolejną zaletą biblioteki iostream jest łatwość modyfikowania działania funkcji obsługujących wejście/wyjście. Szczególnie cenna jest możliwość dowolnego modyfikowania działania operatora wstawiania do strumienia (przeciążanie operatora **<<**).

Operacje wejścia/wyjścia języka C++

```
//Wyświetlanie danych, agregaty danych
#include <iostream.h>
class Data
{  public:
        int d;
        char m[15];
};
int main()
{ Data d1 = {3,"Maj"};
  cout << d1.d << "  " << d1.m << endl;
    getchar();
    return 0;
}
```

Klasa **Data** zawiera dwie publiczne dane składowe. Obiekt klasy **Data** zainicjalizowano za pomocą listy wartości poszczególnych składowych (podobnie jak inicjalizuje się struktury):

```
Data d1 = { 3, "Maj" } ;
```

Taka metoda inicjalizacji może być stosowana do klas będącymi agregatami danych.

Operacje wejścia/wyjścia języka C++

Klasa jest agregatem danych, gdy spełnia następujące warunki:

- wszystkie składowe są publiczne
- nie zdefiniowano konstruktorów
- nie dziedziczy z innej klasy
- nie ma metod wirtualnych

Wartości poszczególnych danych możemy pokazać w klasyczny sposób:

```
cout << d1.d << "  " << d1.m << endl;
```

Musimy zastosować operator dostępu do każdej składowej, za każdym razem musimy pisać pełną nazwę danej.

Operacje wejścia/wyjścia języka C++

Dużym udogodnieniem byłaby możliwość zastosowania instrukcji:

```
cout << d1;
```

Taka instrukcja powinna wypisać wartości wszystkich danych składowych. W języku C++ mamy oczywiście możliwość opracowania takiej instrukcji – wykorzystamy ***przeciążenie operatora*** wstawiania **<<**.

W pliku nagłówkowym **<iostream.h>** znajduje się deklaracja klasy strumieni wyjściowych **ostream** oraz szereg definicji przeciążających operator '**<<**', w których pierwszym jego argumentem jest obiekt klasy **ostream**. Możliwe jest takie przedefiniowanie tego operatora, aby możliwe było wyświetlenie danych typów zdefiniowanych przez użytkownika. W kolejnym przykładzie wykorzystamy możliwość przeciążania operatora wstawiania do strumienia.

Operacje wejścia/wyjścia języka C++

```
// Pokazanie danych, agregaty danych, przeciążony operator <<
#include <iostream.h>
class Data
{ public:
      int d;
      char m[15];
      friend ostream& operator << (ostream&, Data&);
};
 ostream& operator << (ostream& wy, Data& dat)
 { wy << dat.d << "  " << dat.m << endl;
   return wy;
 }

int main()
{ Data d1 = {3,"Maj"};
  cout << d1;
      getchar();
      return 0;
}
```

Operacje wejścia/wyjścia języka C++

Funkcja **operator<<()** nie może być funkcją składową klasy, na których obiektach ma pracować. Ten wymóg bierze się ze sposobu definicji funkcji w **<iostream.h>.** Gdyby funkcja operatorowa była funkcją składową, to jej pierwszym niejawnym argumentem (przekazywanym przez wskaźnik **this**) byłby obiekt, wywołujący tą funkcję. Zgodnie z definicją, pierwszym z lewej argumentem musi być strumień klasy **ostream,** natomiast prawym operandem jest obiekt, który chcemy wyprowadzić na standardowe wyjście.

Deklaracja funkcji operatorowej ma postać:

```
friend ostream& operator << (ostream&, Data&);
```

a definicja tej funkcji jest następująca:

```
ostream& operator << (ostream& wy, Data& dat )
{
    wy << dat.d << "  " << dat.m << endl;
    return wy;
 }
```

Operacje wejścia/wyjścia języka C++

Funkcja operatorowa pobiera jako argumenty referencję do **ostream** (**wy**) oraz **da**t do zdefiniowanego uprzednio typu (klasy) **Data**. Rezultatem wykonania funkcji jest zwracana referencja do **ostream**. Funkcja **operator <<** wyświetla obiekty typu **Data**.

Gdy kompilator napotka wyrażenie

cout << d1;

przekształci je na wywołanie funkcji

operator << (cout, d1) ;

Funkcja operatorowa została zadeklarowana jako **friend**, także z tego powodu, że w ogólności chcemy mieć dostęp do danych składowych klasy, które nie są publiczne (w naszym przykładzie składowe **d** i **m** są publiczne).

Operacje wejścia/wyjścia języka C++

```
//Pokazanie danych, dane prywatne, przeciążony operator <<
#include <iostream.h>
#include <string.h>

class Data
{     int d;
      char m[15];
  public:
      Data(int d, char* m);
      friend ostream& operator << (ostream&, Data&);
};

 Data::Data(int d, char* m)        //konstruktor
 {  strcpy ( this -> m, m ) ;
    this  -> d = d ;
 }

 ostream& operator << (ostream& wy, Data& dat)
 { wy << dat.d << "  " << dat.m << endl;
   return wy;
 }

int main()
{  Data d1 = { Data (1, "Maj") };
   Data d2 (3, "Maj");
   cout << d1 << d2;
     getchar();
     return 0;
}
```

Operacje wejścia/wyjścia języka C++

Aby zainicjalizować obiekt klasy **Data** użyjemy konstruktora:

```
Data::Data(int d, char* m)        //konstruktor
 {  strcpy ( this -> m, m ) ;
    this  -> d = d ;
}
```

Przypominamy, że napis w języku C traktowany jest jak tablica znaków zawierająca znak NUL na końcu (znacznik końca napisu, literałem tego znaku jest '\0'). Zbiór funkcji działających na C-napisach umieszczony jest w pliku nagłówkowym <string.h> .

Funkcja **strcpy()** operuje na tzw. C-napisach (w języku C++ możemy posługiwać się napisami, tak jak w języku C). Prototyp tej funkcji ma postać:

```
char *strcpy ( char *dest, const char *src ) ;
```

Operacje wejścia/wyjścia języka C++

Funkcja kopiuje łańcuch **src** do łańcucha **dest**, przerywając tę operację po osiągnięciu znaku końca (**'\0'**) łańcucha **src**. Znak ten jest przenoszony do łańcucha **dest.** Wartością funkcji jest **dest**. Przypominamy również, że każda funkcja składowa klasy ma zdefiniowany niejawny stały wskaźnik o nazwie **this**. Gdy funkcja składowa jest wywoływana, wskaźnik ten zostaje zainicjalizowany na adres egzemplarza klasy, dla której ta funkcja została wywołana.

Obiekty klasy **Data** tworzone są na dwa sposoby:

```
Data d1 = { Data (1, "Maj") };
Data d2 (3, "Maj");
```

Na koniec pokażemy zastosowanie przeciążonego operatora wstawiania do strumienia do wyświetlania danych z tablicy obiektów.

Operacje wejścia/wyjścia języka C++

```
// Tablica obiektów, przeciążony operator <<
#include <iostream.h>
#include <string.h>

class Data
{     int d;
      char * m;
  public:
    Data (int dd, char* mm)
            : d(dd), m(strcpy(new char[strlen(mm)+1],mm)) {  }
    friend ostream& operator << (ostream&, Data&);
};

 ostream& operator << (ostream& wy, Data& dat)
 { wy << dat.d << "  " << dat.m << endl;
   return wy;
 }

int main()
{  Data dm[] = { Data (1, "Maj"), Data(3, "Maj"), Data (5, "Maj") };
   for (int i = 0; i <3; i++)   cout  <<   dm[i] ;
       getchar();
       return 0;
}
```

Operacje wejścia/wyjścia języka C++

Tablica obiektów jest utworzona przy pomocy listy inicjującej tablicy. W nawiasach klamrowych wywołano jawnie konstruktory:

```
Data dm[ ] = { Data (1, "Maj"), Data(3, "Maj"), Data (5, "Maj") };
```

Konstruktor ma postać:

```
Data (int dd, char* mm)
    : d(dd), m ( strcpy (new char [strlen (mm)+1], mm ) )  {  }
```

Wewnątrz klasy **Data** deklarujemy zaprzyjaźnioną funkcję przeciążającą operator wstawianie do strumienia dla obiektów klasy **Data**. Funkcja operatorowa ma bezpośredni dostęp do danych prywatnych, w pętli **for** wyświetlane są te dane:

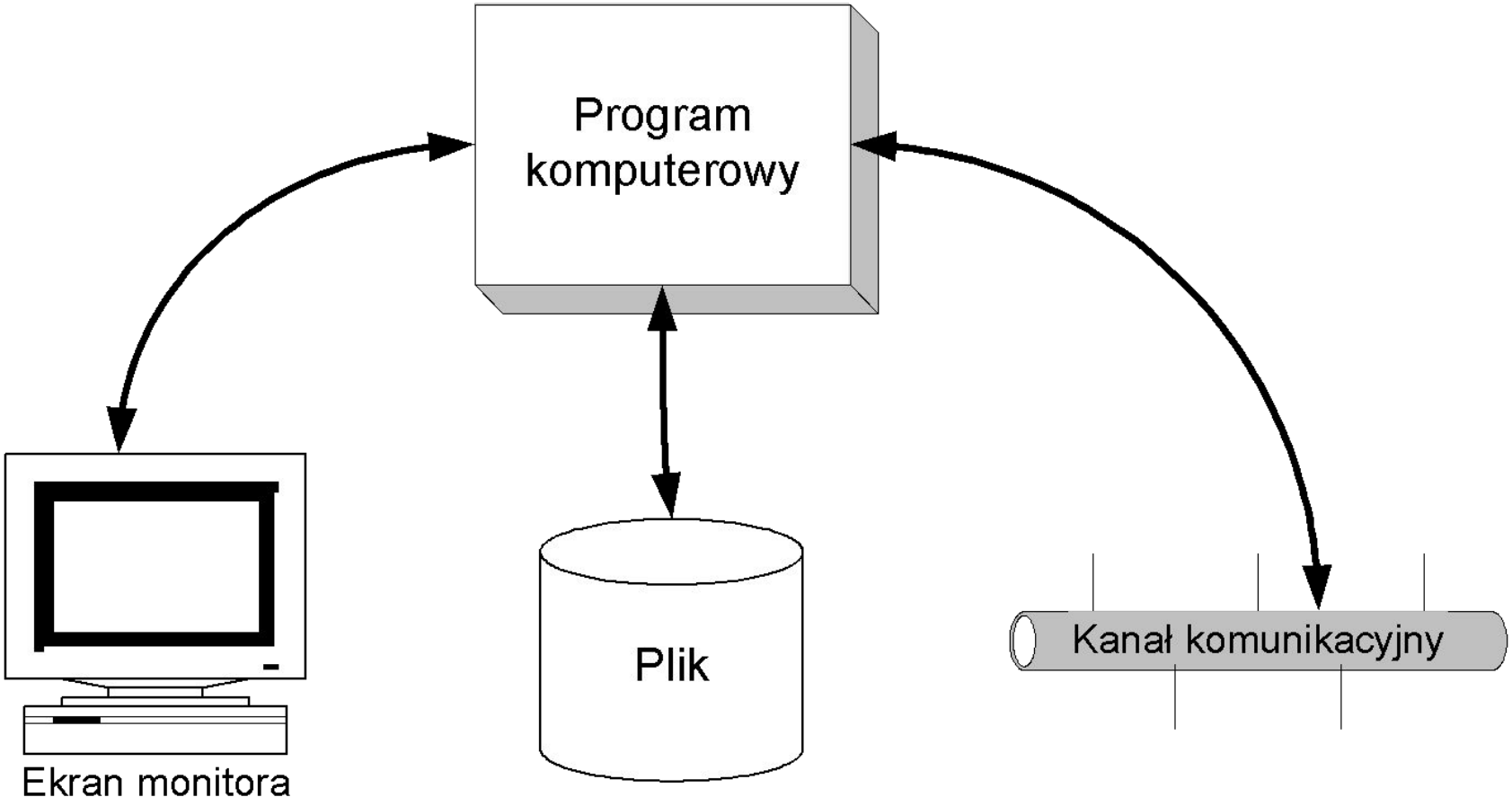
1 Maj
3 Maj
5 Maj

Operacje wejścia/wyjścia języka C++

W języku C++ mamy do czynienie z całkiem innymi technikami obsługiwania operacji wejścia/wyjścia, aczkolwiek te operacje, podobnie jak w języku C, odbywają się za pośrednictwem strumieni. Przypominamy, że ***strumień*** to urządzenie logiczne produkujące lub pobierające informacje. Strumień jest połączony z urządzeniem fizycznym (klawiatura, monitor, itp.) poprzez system wejścia/wyjścia. Koncepcja strumienia jest bardzo pożyteczna, pozwala na jednolite traktowanie urządzeń tak różnych jak ekran monitora, drukarka czy dyskietka. Ta sama funkcja może być wykorzystana do zapisu danych na dyskietce lub do wyświetlenia danych na ekranie.

Operacje wejścia/wyjścia języka C++

Transport danych wspierany obsługiwany przez bibliotekę IOStreams



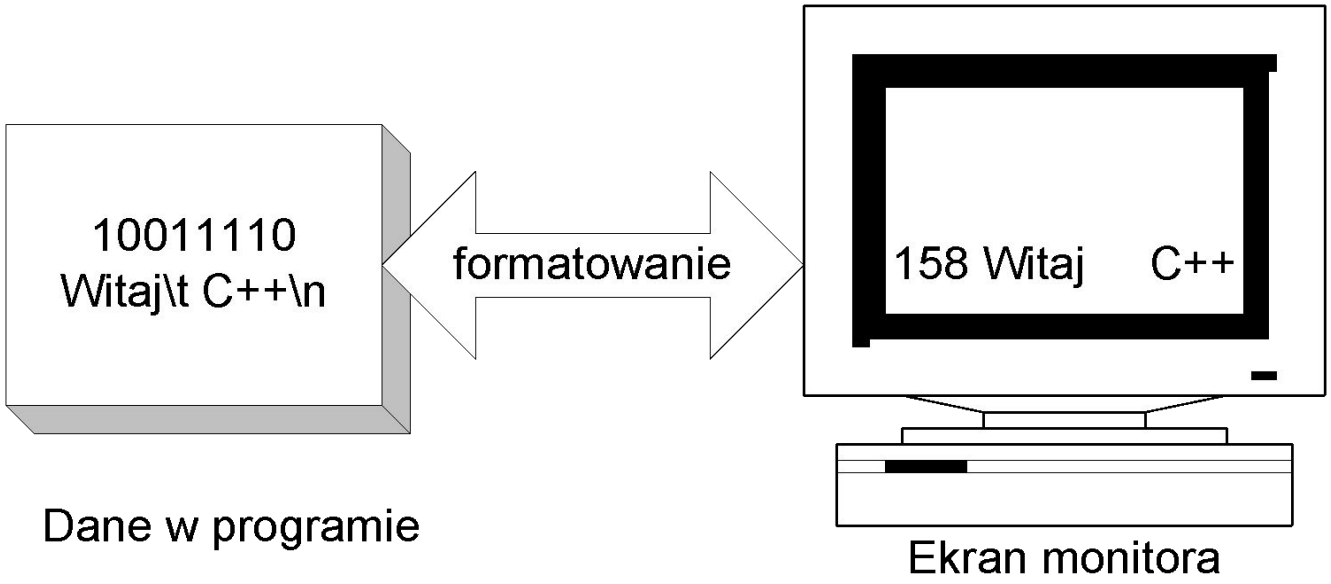
Operacje wejścia/wyjścia języka C++

Biblioteka IOStreams implementuje mechanizm wejścia-wyjścia w bibliotece standardowej (ANSI/ISO) języka C++. Zasadniczo wejście i wyjście to transfer danych pomiędzy programem a dowolnym urządzeniem zewnętrznym, tak jak to pokazano na rysunku . Transport danych może odbywać się w sposób strumieniowy lub blokowy. Biblioteka IOStreams (jak wskazuje nazwa) wspiera strumieniowe operacje wejścia – wyjścia. W tego typu organizacji przesyłania danych mamy do czynienia ze strumieniem bajtów, znaków lub innych jednostek informacji o ustalonym i równym rozmiarze. Dane przepływają pomiędzy programem i urządzeniami zewnętrznymi. Praktycznie użytkownik biblioteki IOStreams ma do czynienia ze ***strumieniem znaków***, biblioteka jest ukierunkowana na ***tekstowe operacje*** wejścia – wyjścia. Reprezentacja danych w programie może się różnić od reprezentacji danych w urządzeniach zewnętrznych. Mówimy o ***reprezentacji wewnętrznej*** (np. wartości całkowite w programie są przechowywane w postaci binarnej) oraz o ***reprezentacji zewnętrznej*** (dane wyświetlane na ekranie mają postać cyfr lub liter). W zależności od rodzaju reprezentacji zewnętrznej operacje wejścia – wyjścia dzielimy na ***tekstowe*** i ***binarne***.

Operacje wejścia/wyjścia języka C++

W dużej części IOStreams jest używany do przetwarzania tekstu. Przetwarzanie teksu wymaga przeprowadzenia dwóch operacji: formatowania (ang. *formatting*) i konwersji kodu (ang. *code conversion*).

Formatowanie polega na transformacji sekwencji bajtów reprezentujących wewnętrzne dane do zrozumiałej przez człowieka sekwencji znaków, np. zmiana wartości całkowitej przechowywanej w zmiennej do sekwencji cyfr (rysunek).



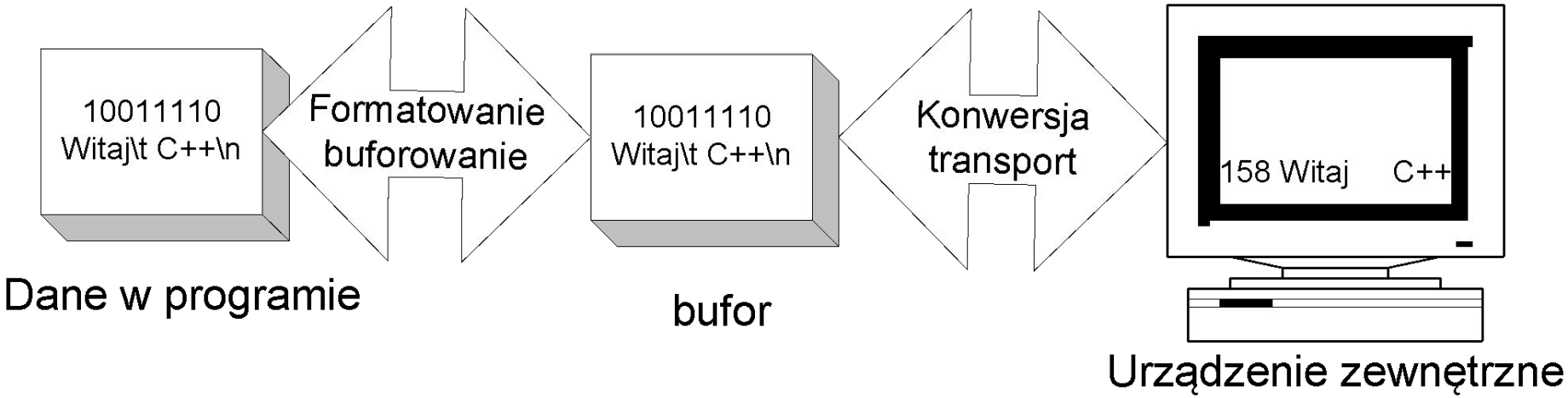
Formatowanie danych programu

Operacje wejścia/wyjścia języka C++

Konwersja kodu (ang. *code conversion*) polega na translacji reprezentacji znaku. Jak wiadomo w C++ mamy możliwość używania znaków wielobajtowych (ang. multibyte characters, wide characters). Często zachodzi potrzeba translacji znaków wielobajtowych przechowywanych wewnętrznie (w programie) do sekwencji znaków wyświetlanych np. na ekranie. Przesyłanie danych może odbywać się za pośrednictwem buforów, w związku z czym operacje wprowadzania i wyprowadzania danych realizowane przy pomocy biblioteki IOStreams wymagają w najogólniejszym przypadku czterech etapów (rysunek):

- formatowanie na wyjściu i analizę leksykalną na wejściu
- buforowanie
- konwersja kodu
- transport

Operacje wejścia/wyjścia języka C++



Etapy wprowadzania i wyprowadzania strumieni danych w IOStreams

Operacje wejścia/wyjścia języka C++

Na I etapie wyprowadzania (wprowadzania) strumieni danych mamy do czynienia z ***translacją*** pomiędzy ciągiem bajtów tworzących wewnętrzną reprezentacje danych w programie a wyjściowym ciągiem znaków (operacje te nazywamy formatowaniem). Podczas wprowadzania i wyprowadzania strumienia danych musimy dokonać także analizy leksykalnej (ang. *parsing*) na wejściu.

Na tym etapie wykonywanych jest szereg operacji – część operacji wykonywanych jest automatycznie, część operacji nadzoruje użytkownik.

Najważniejsze operacje to:

- ustalanie precyzji (cyfr znaczących po kropce dziesiętnej) i typu reprezentacji wartości zmiennoprzecinkowej
- wybór systemu liczbowego dla strumienia danych : szesnastkowy, dziesiętny, ósemkowy. IOStreams ma możliwość generowania także reprezentacji liczb o innych podstawach
- Usuwanie znaków niedrukowalnych (ang. white space) ze strumienia wejściowego
- Ustalanie szerokości pola, w jakim będą umieszczane wartości wyprowadzane na wyjściu
- Wykonywanie wielu innych operacji (np. wybór kropki lub przecinka w reprezentacji liczb na wyjściu)

Operacje wejścia/wyjścia języka C++

- Etap II, czyli buforowanie jest procesem opcjonalnym. W bibliotece IOStreams operacje strumieniowe są buforowane domyślnie, ale na żądanie buforowanie może być wyłączone. ***Buforowanie*** oznacza proces przechowywania sekwencji znaków w pamięci pomiędzy etapem formatowania (także podczas analizy leksykalnej) i etapem transportu do urządzenia zewnętrznego. Buforowanie stosowane jest najczęściej w celu zmniejszenia całkowitego czasu transmisji danych.
- W etapie III, w którym mamy do czynienia z ***konwersją kodu***, wykonywana jest translacja reprezentacji znaków. Na przykład w celu zapewnienia obsługi znaków narodowych (polskich, japońskich, itp.) język C++ uwzględnia tzw. ***internacjonalizację,*** wprowadzając w tym celu wielobajtową reprezentacje znaków (w odróżnieniu od znanych kodów ASCII, gdzie znak kodowany jest na jednym bajcie). Zazwyczaj znaki w reprezentacji wewnętrznej przechowywane są jako tzw. szerokie znaki (ang. *wide characters*), a reprezentacja zewnętrzna (znaki zapisywane w plikach) przechowuje znaki w postaci wielobajtowej. Znaki wielobajtowe mają różne rozmiary – mogą być jednobajtowe, najczęściej są dwubajtowe, ale mogą wymagać także więcej niż dwóch bajtów. W takim przypadku translacja jest niezbędna podczas przesyłania znaków. Wszystkie szerokie znaki mają ten sam rozmiar, są najczęściej stosowane do wewnętrznego przetwarzania. Wielobajtowe znaki maja różne rozmiary i są przechowywane w bardziej upakowanej postaci. Typowo są używane do transferu danych lub do przechowywania w urządzeniach zewnętrznych, jakim np. są pliki.

Operacje wejścia/wyjścia języka C++

- W etapie IV mamy do czynienia z ***transportem znaków*** z programu do urządzenia zewnętrznego lub odwrotnie. Na tym etapie system musi wykonać wiele operacji:
- Aby przesyłać dane, musimy nawiązać połączenie pomiędzy programem i urządzeniem zewnętrznym, na tym etapie musimy dysponować wiedzą na temat sposobu nawiązywania takiego połączenia, stąd mamy np. operacje otwierania i zamykania plików.
- Aby zredukować liczbę połączeń z urządzeniem zewnętrznym możemy użyć buforów. Dane wyjściowe wysyłane są podczas transportu do bufora strumienia (ang. *stream buffer*). Kolejny transport danych do zewnętrznego urządzenia nastąpi wtedy, gdy bufor jest pełny. W przypadku danych wejściowych, dane są odczytywane z urządzenia zewnętrznego i zapełniany jest bufor. Program otrzymuje dane z bufora. Gdy bufor jest pusty, system musi ponownie zapełnić bufor.

Klasy strumieni

Biblioteka IOStreams zawiera rozbudowane klasy do obsługi różnorodnych operacji wejścia – wyjścia. Klasy te nazywane są klasami strumieni. Między klasami strumieni istnieją dość skomplikowane powiązania (mamy klasy bazowe i klasy pochodne, dziedziczenie może być proste i wielobazowe).

Biblioteka IOStreams zawiera:

- Dwie bazowe klasy strumieni (ios_base i basic_ios)
- Ogólne klasy wejścia i wyjścia strumieni (basic_istream, basic_ostream, basic_iostream)
- Klasy strumieni plikowych (basic_ifstream, basic_fstream, basic_ofstream)
- Klasy ciągów znakowych (basic_istringstream, basic_stringstream, basic_ostringstream)

Klasy strumieni

Definicje klas strumieni zostały umieszczone w kilku różnych plikach nagłówkowych. W praktyce w celu obsługi operacji wejścia/wyjścia najczęściej włączane są do programów trzy pliki nagłówkowe:

<iostream>

Plik zawiera definicje globalnych obiektów strumieni takich jak cin, cout, cerr, clog, wcin, wcout, wcerr i wclog

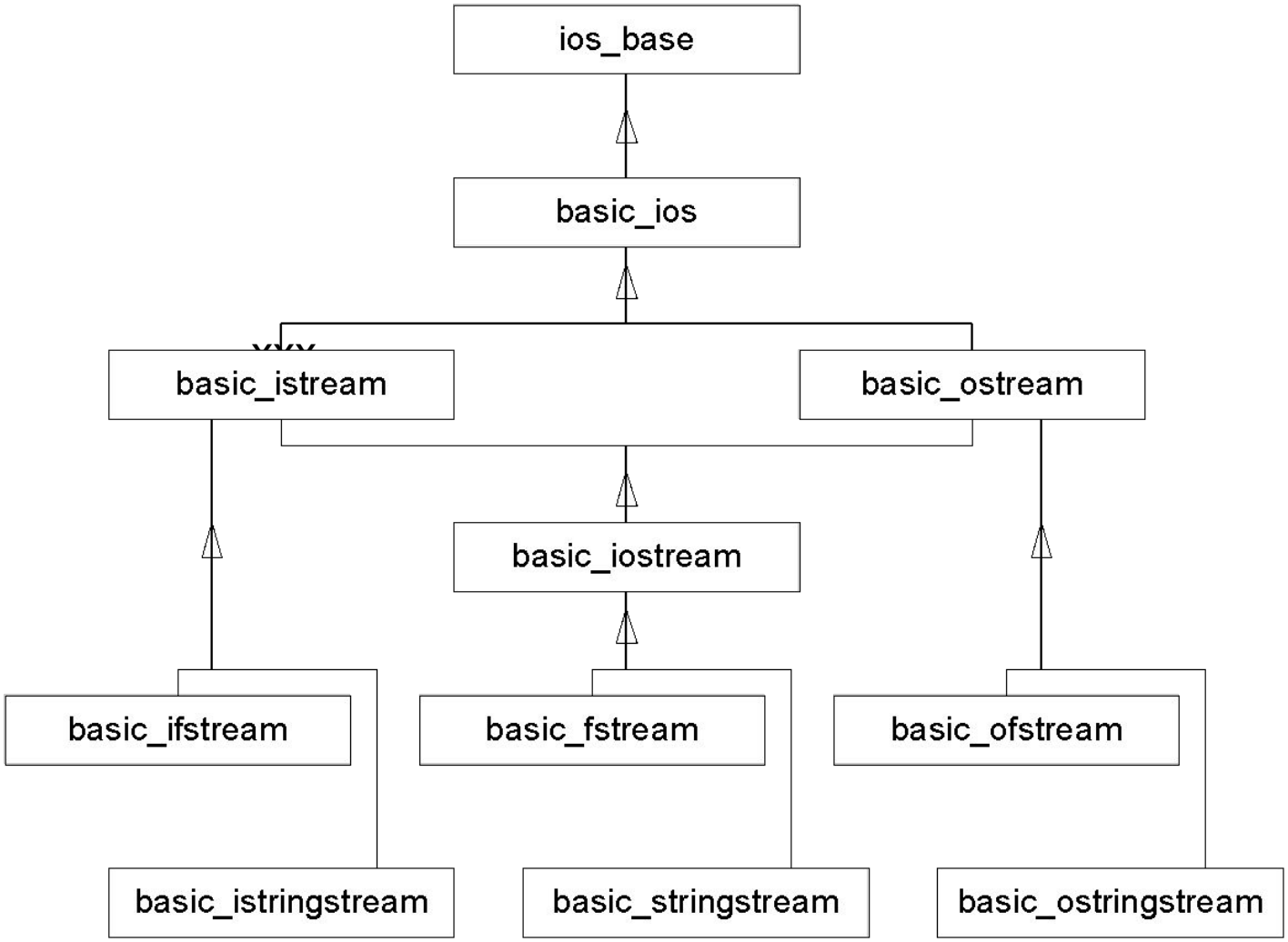
<iomanip>

Plik zawiera definicje standardowych manipulatorów strumieni. Są tam umieszczone szablony funkcji resetioflags, setioflags, setbase, setfill, setprecision i setw.

<fstream>

Plik zawiera deklaracje klas strumieni plikowych, należy włączać ten plik zawsze, gdy użytkownik wykorzystuje operacje przetwarzania plików

Klasy strumieni biblioteki IOStreams



Klasy strumieni

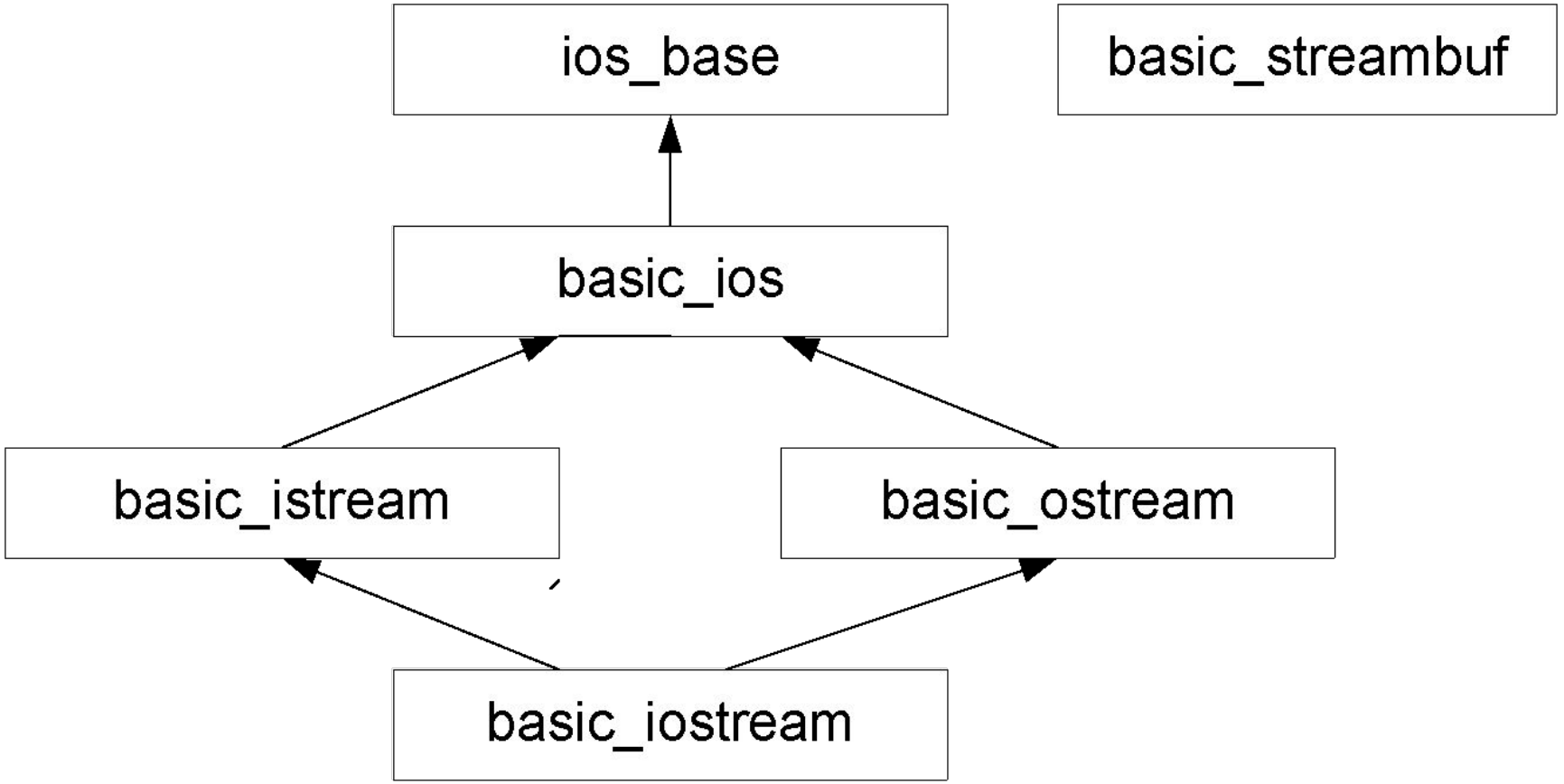
Biblioteka IOStreams jest bardzo rozbudowana, pełniejszy zestaw plików nagłówkowych zawierających elementy tej biblioteki pokazano poniżej.

<fstream> Zawiera deklaracje klas strumieni plikowych
<iomanip> Zawiera deklaracje manipulatorów strumieni
<ios> Zawiera deklaracje klas bazowych strumieni
<iosfwd> Zawiera deklaracje zapowiadające klas strumieni
<iostream> Zawiera deklaracje globalnych obiektów strumieni
<istream> Zawiera deklaracje klas strumieni wejściowych
<ostream> Zawiera deklaracje klas strumieni wyjściowych
<sstream> Zawiera deklaracje klas strumieni ciągów znaków
<streambuf> Zawiera deklaracje szablonu klasy bazowej buforów strumieni

Klasy strumieni

W hierarchii klas podstawową rolę odgrywa klasa bazowa o nazwie ios_base. Klasa bazowa ios_base określa składowe wszystkich klas strumieniowych. Wzorzec klasy basic_ios dziedziczy z klasy ios_base i definiuje wspólne składowe wszystkich klas strumieni. We wzorcu znajduje się definicja bufora używanego przez strumień. Bufor ten jest obiektem klasy dziedziczącej z klasy wzorca o nazwie basic_streambuf. Wzorce klas basic_istream oraz basic_ostream dziedziczą w sposób wirtualny z klasy basic_ios (rysunek).

Hierarchia klas podstawowych klas strumieniowych biblioteki IOStreams



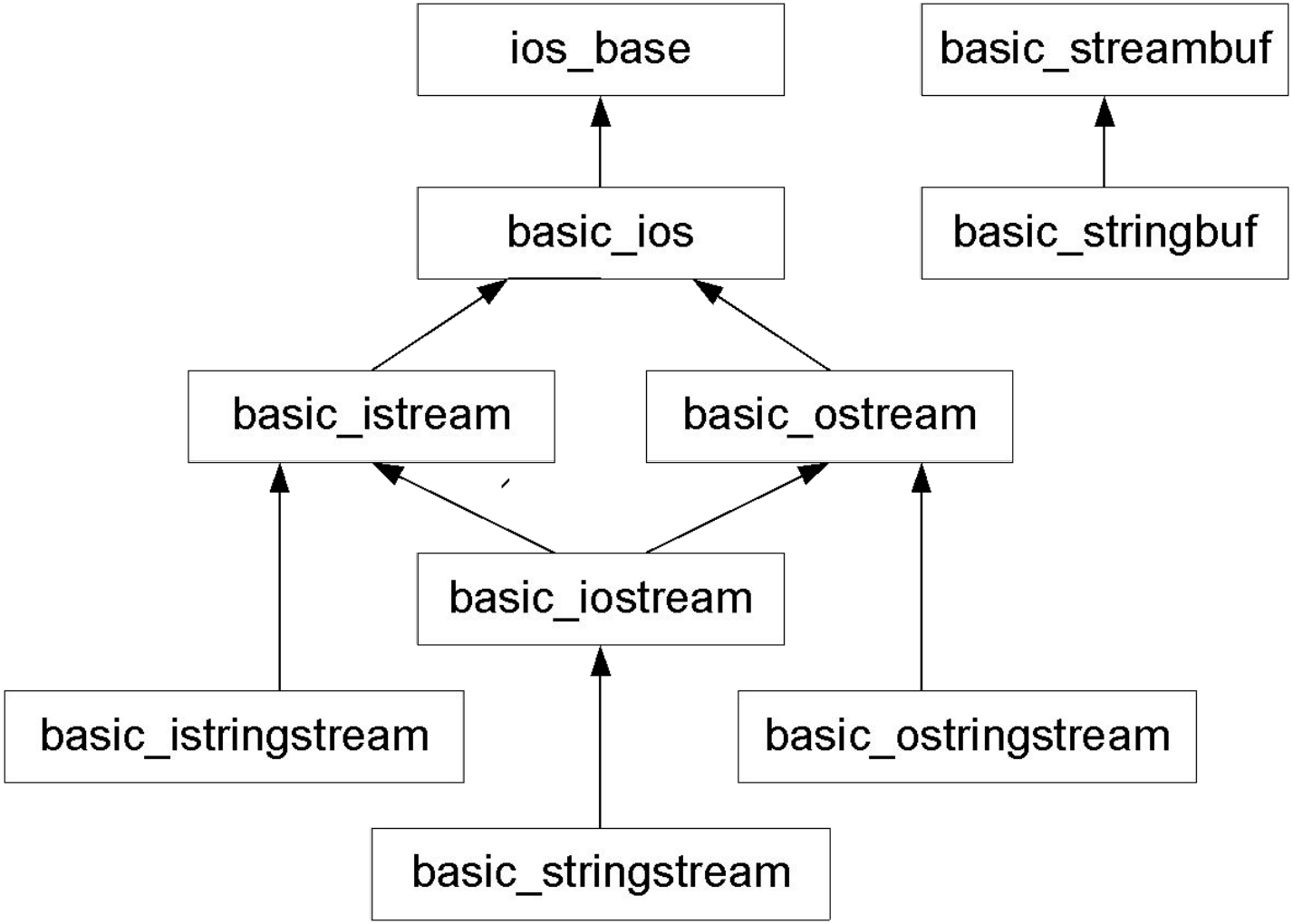
Klasy strumieni

- Klasa basic_istream definiuje obiekty wykorzystywane w operacjach wejściowych. W tej klasie zdefiniowany jest przeciążony operator „>>” (standardowy operator strumienia, zwany z angielska ***insterterem***) oraz obiekt „cin” reprezentujący standardowy strumień wejściowy.
- Klasa basic_ostream definiuje obiekty wykorzystywane w operacjach wyjściowych. W tej klasie zdefiniowany jest przeciążony operator „<<” (standardowy operator strumienia, zwany z angielska ***ekstraktorem***) oraz obiekt „cout” reprezentujący standardowy strumień wyjściowy a także obiekty cerr i clog, łącznie z manipulatorami endl i ends.
- Klasa basic_iostream dziedziczy zarówno z klasy basic_istream i klasy basic_ostream. Z tego wynika, że korzystając z klasy basic_iostream mamy możliwość jednoczesnego obsługiwania strumieni wejściowych i strumieni wyjściowych. Ten wzorzec klasy definiuje obiekty, które mogą być wykorzystane w celu zarówno odczytu, jak też zapisu.

Klasy strumieni

- Odczytywanie i zapisywanie łańcuchów znakowych przy pomocy biblioteki IOStreams jest zagadnieniem dość skomplikowanym.
- W starej bibliotece do reprezentacji łańcuchów znakowych używany był typ char*, w nowej, aktualnej i zalecanej bibliotece używany jest „***typ***” string. Stare klasy strumieni dla łańcuchów znakowych stanowią również część biblioteki standardowej C++. Mamy dwa kompletnie różne sposoby obsługi łańcuchów znakowych, co prowadzi do wielu nieporozumień i kłopotów. Zaleca się, aby nie używać typu char* do reprezentacji łańcuchów znakowych, gdyż w przyszłości te stare klasy będą usunięte ze standardu (od 10 lat tego nie zrobiono?).
- Mechanizmy udostępniane przez klasy strumieni pozwalają na wykonywanie operacji odczytu i zapisu łańcuchów znakowych przez udostępnienie bufora, ale nie posiadają kanału wejścia/wyjścia. Zawartość bufora może być przetwarzana przy pomocy specjalnych funkcji.

Hierarchia klas strumieniowych biblioteki IOStreams do obsługi łańcuchów znakowych



Klasy strumieni

- Klasa basic_istringstream dziedziczy z klasy basic_istream i zapewnia odczyt danych z łańcuchów znakowych. Źródłem dla klasy basic_istringstream jest obiekt klasy string, czyli napis. Klasa basic_istringstream wymaga dołączenia pliku <sstream>.
- Klasa basic_ostringstream dziedziczy z klasy basic_ostream i zapewnia zapis danych do łańcuchów znakowych. Ujściem dla klasy basic_ostringstream jest obiekt klasy string, czyli napis. Klasa basic_ostringstream wymaga dołączenia pliku <sstream>.
- Klasa basic_stringstream dziedziczy z klasy basic_iostream i zapewnia zapis danych do łańcuchów znakowych oraz odczyt danych z łańcuchów znakowych. Wymaga obiektów klasy string.
- Klasa wzorcowa basic_stringbuf wykorzystywana jest do właściwej operacji odczytu i zapisu pojedynczego znaku (w bibliotece wejścia/wyjścia klasy pochodne klasy basic_ios służą jedynie do formatowania danych).

Klasy strumieni

Jak już wspominaliśmy, w bibliotece IOStream mamy dodatkowy zestaw klas strumieni dla wartości typu char*. Te klasy strumieni umożliwiają formatowany dostęp do C-łańcuchów (pierwotnie zdefiniowanych w języku C) i udostępniane są jedynie w celu zachowania zgodności z wczesnymi wersjami języka C++. Mamy następujące klasy strumieni obsługujące C-łańcuchy:

- klasa istrstream obsługuje odczyt sekwencji znaków
- klasa ostrstream obsługuje zapis sekwencji znaków
- klasa strstream obsługuje zapis i odczyt sekwencji znaków
- klasa strstreambuf jest używana w charakterze bufora strumienia dla wartości typu char*

Wymienione klasy obsługujące strumienie dla wartości typu char* wymagają włączenia pliku nagłówkowego <strstream>. Strumień klasy istrstream może zostać zainicjowany na dwa sposoby: C-łańcuch musi być zakończony znakiem końca łańcucha '\0' lub długość łańcucha musi zostać przekazana jako dodatkowy parametr.

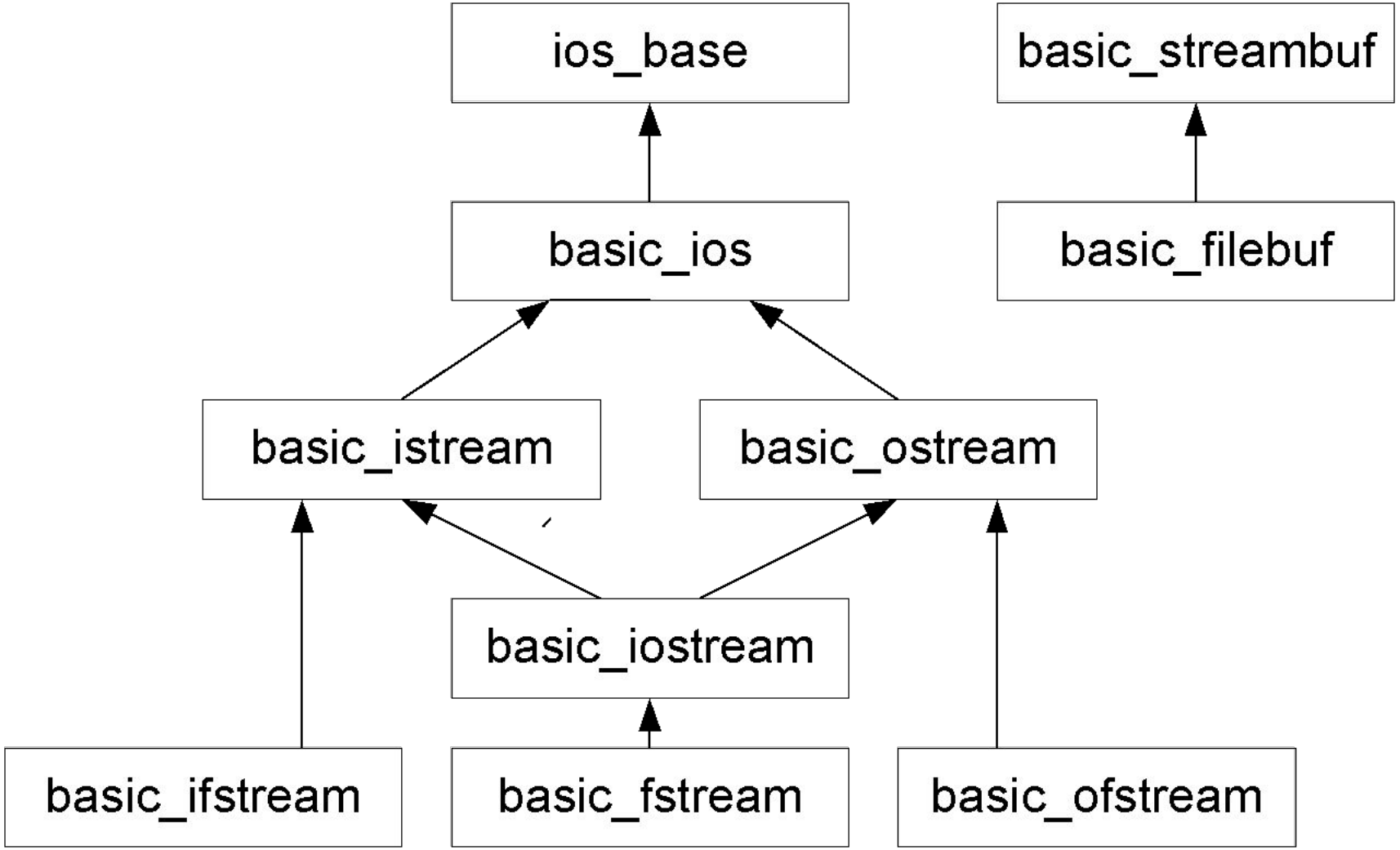
Plikowe operacje wejścia/wyjścia języka C++

Biblioteka IOStreams zawiera odpowiednie klasy obsługujące operacje wejścia/wyjścia na plikach. Hierarchia plikowych klas strumieni pokazana jest na rysunku.

Mamy do dyspozycji cztery wzorce klas:

- klasa wzorcowa basic_ifstream służy do odczytu danych z pliku
- klasa wzorcowa basic_ofstream służy do zapisu danych do pliku
- klasa wzorcowa basic_fstream służy do odczytu danych z pliku oraz do zapisu danych do pliku
- klasa wzorcowa basic_filebuf używana jest przez inne klasy strumieni plikowych do operacji odczytu i zapisu znaków do pliku

Hierarchia klas strumieniowych biblioteki IOStreams do obsługi operacji wejścia/wyjścia na plikach



Plikowe operacje wejścia/wyjścia języka C++

W bibliotece standardowej zdefiniowano osiem globalnych obiektów strumieni. Dzięki tym obiektom można uzyskać dostęp do standardowych kanałów wejścia/wyjścia. Oprócz obiektów strumieni obsługujących zwykła znaki typu char (cin, cout, itp.) mamy do dyspozycji także obiekty dla standardowych kanałów strumieni wykorzystujących typ wchar_t. Obiekty globalne strumieni pokazane są w tabeli .

Nazwa	Typ	Opis
cin	istream	Standardowy kanał wejścia. Odczytuje dane wejściowe, najczęściej z klawiatury
cout	ostream	Standardowy kanał wyjścia. Zapisuje dane wyjściowe, najczęściej na ekranie
cerr	ostream	Standardowy kanał wyjścia błędów. Zapisuje komunikat o błędach najczęściej na ekranie, nie jest buforowany.
clog	ostream	Standardowy kanał wyjścia dziennika (logów). Rzadko używany, jest buforowany. Zapis najczęściej na ekranie.
wcin	wistream	Standardowy kanał wejścia. Odczytuje „szerokie” dane wejściowe, najczęściej z klawiatury
wcout	wostream	Standardowy kanał wyjścia. Zapisuje „szerokie” dane wyjściowe, najczęściej na ekranie
wcerr	wostream	Standardowy kanał wyjścia błędów. Zapisuje „szerokie” komunikaty o błędach najczęściej na ekranie, nie jest buforowany.
wclog	wostream	Standardowy kanał wyjścia dziennika (logów). Rzadko używany, „szerokie znaki”, jest buforowany. Zapis najczęściej na ekranie.

Plikowe operacje wejścia/wyjścia języka C++

W nowej bibliotece wejścia/wyjścia globalne obiekty strumieni zadeklarowane są w przestrzeni nazw **std**:

```
std :: istream cin;
std :: ostream cout;
std :: ostream cerr;
```

Wobec tego, aby wykorzystać jakąś składową biblioteki nowego typu, musimy podać nazwę tej składowej poprzedzoną nazwą obszaru ***namespace*** i operatorem zasięgu.

Instrukcja pozwalająca na wyprowadzenie napisu na ekran może mieć następującą postać:

```
std::cout << "Hello, World";
```

Poprzedzanie nazwy każdego obiektu identyfikatorem przestrzeni nazw może być uciążliwe, przy pomocy słowa kluczowego ***using*** można obejść tą niedogodność. Umieszczenie na początku pliku kodu źródłowego instrukcji:

```
using namespace std;
```

udostępnia przestrzeń nazw ***std***, elementy obszaru ***namespace std*** mogą być używane w pliku bez konieczności poprzedzania każdego z nich nazwą.

Plikowe operacje wejścia/wyjścia języka C++

- Ze wszystkich tych strumieni można korzystać przy pomocy przeciążonych operatorów „<<” oraz „>>” a także odpowiednich funkcji składowych klas. Informacje o stanie strumienia reprezentowane są przy pomocy ***stałych bitowych*** wprowadzonych w klasie ios_base. Dla strumieni zdefiniowano różne ***manipulatory***. Manipulatory są specjalnymi obiektami, których zadaniem jest formatowanie zawartości strumieni lub opróżnianie jego buforów.
- Oprócz standardowych operatorów strumieni (<< i >>) standardowa biblioteka IOStreams udostępnia funkcje składowe klas strumieni, które także umożliwiają realizację odczytu i zapisu danych. W kolejnym przykładzie zilustrujemy podstawowe aspekty użycia przeciążonych operatorów << i >>.

Plikowe operacje wejścia/wyjścia języka C++

```
// Przeciążone operatory wejścia/wyjścia << i >>
#include <iostream>     //dla klas strumieni
#include <cmath>        //dla funkcji sqrt()
#include <cstdlib>      //dla stalej EXIT_FAILURE
using namespace std;  //zapobiega pisaniu std::cout
int main()
{
 double x;
 cout << "Pierwiastek kwadratowy z liczby \n" ;
 cout << " x = ";
 if (!(cin >> x))
        {  cerr << "Zly odczyt x " << endl;
           getchar();
           return EXIT_FAILURE;
         }
 if (x <= 0)
        { cerr << " x jest ujemne " << endl;
          getchar();
          return EXIT_FAILURE;
        }
 cout << "Pierwiastek z " << x << " = " << sqrt(x) << endl;
  getchar();     // do "zatrzymania ekranu" w pakiecie Builder C++
  return 0;
}
```

Plikowe operacje wejścia/wyjścia języka C++

Pokazany program wczytuje liczbę, wylicza pierwiastek kwadratowy z tej liczby i wyświetla wynik . Program sprawdza, czy wprowadzona dana z klawiatury ma poprawną wartość. Jeżeli wystąpi błąd odczytu (zostanie wprowadzona np. litera zamiast liczby) to program wyświetla komunikat o błędzie korzystając ze strumienia wyjścia błędów (obiekt cerr klasy ostream) i kończy działanie zwracając status błędu (stała EXIT_FAILURE). Podobnie zachowa się program w przypadku wprowadzenia z klawiatury ujemnej wartości.

Definicje klas strumieniowych rozmieszczone są w kilku plikach nagłówkowych.

Plik nagłówkowy <iostream> zawiera deklaracje globalnych obiektów strumieni (w rodzaju cin, cout, cerr). Instrukcja

```
cout << "Pierwiastek kwadratowy z liczby \n" ;
```

powoduje, że literał znakowy jest zapisany do standardowego kanału wyjścia. Operacja wykonana jest przy pomocy operatora „<<”, wywołanego na rzecz obiektu cout. Użycie operatorów „<<” i „>>” powoduje wywołanie odpowiedniej funkcji operatorowej dwuargumentowej. Po lewej stronie operatora jest umieszczany pierwszy parametr będący odniesieniem do obiektu-strumienia (np. cout), a po prawej stronie operatora występuje drugi parametr, będący wartością wyrażenia.

W pokazanym przykładzie drugim parametrem wywołania operatora „<<” jest literał znakowy (napis umieszczony miedzy znakami cudzysłowu). Operatory wyjścia mogą być przeciążane nie tylko dla typów podstawowych, ale także dla typów definiowanych przez użytkownika.

Obsługa wyjścia – obiekt cout, flagi formatowania

W bibliotece standardowej dane wyjściowe są strumieniem bajtów. Do obsługi tego strumienia najczęściej używany jest obiekt cout łącznie z operatorem <<, nazywanym także operatorem wstawiania.

Operator wstawiania << rozpoznaje wszystkie podstawowe typy języka C++:

- unsigned char
- signed char
- char
- short int (skrótowo : short)
- unsigned short int (skrótowo : unsigned short)
- int
- unsigned int
- long int (skrótowo : long)
- unsigned long int (skrótowo : unsigned long)
- float
- double
- long double

Obsługa wyjścia – obiekt cout, flagi formatowania

Klasa ostream zapewnia definicję funkcji operator<< () także dla następujących typów wskaźnikowych:

- const signed char *
- const unsigned char *
- const char *
- void *

Pamiętamy, że języku C/C++ łańcuch traktowany jest jako tablica przechowująca typ znakowy. Inaczej mówiąc łańcuch reprezentowany może być przez wskaźnik na miejsce położenia tego łańcucha w pamięci, możemy zatem zadeklarować jawny wskaźnik do obsługi łańcucha. Oczywiście w języku C++ wskaźnik może przyjąć postać nazwy tablicy typu char. Łańcuch możemy obsługiwać także przy pomocy napisu ujętego w cudzysłów.

Obsługa wyjścia – obiekt cout, flagi formatowania

W kolejnym przykładzie demonstrujemy instrukcje cout wyświetlające łańcuchy.

```
//Przeciążone operatory wyjścia << i wskaźniki
#include <iostream>
#include <conio.h>
int main()
{
  using std :: cout;
  char nazwisko[30] = "Kowalski";   //tablica typu char
  char *imie = "Jan";               //jawny wskaznik na typ char
  cout << "Obywatel ";              // napis
  cout << imie;
  cout << " ";
  cout << nazwisko;
  getche();
  return 0;
  }
```

Wszystkie pokazane w przykładzie instrukcje cout wyświetlają łańcuchy.

Obsługa wyjścia – obiekt cout, flagi formatowania

Bardziej skomplikowane jest zagadnienie wyświetlania adresu łańcucha

```
//Przeciążone operatory wyjścia << i adresy
#include <iostream>
#include <conio.h>
int main()
{
  using std :: cout;
  using std :: endl;
  int ile_kilo = 1000;
  char *waga = " tona ";
  cout << "adres zmiennej typu int = "<< &ile_kilo << endl;
  cout << "adres zmiennej wskazywanej = " << (void *) waga;
  getche();
  return 0;
}
```

Obsługa wyjścia – obiekt cout, flagi formatowania

Jeżeli musimy wyświetlić adres łańcucha musimy wykonać rzutowanie na typ void.

W wyniku wykonania tego programu otrzymujemy komunikat:

adres zmiennej typu int = 1245064

adres zmiennej wskazywanej = 4202660

W instrukcji:

```
cout << "adres zmiennej wskazywanej = " << (void *) waga;
```

wykonano rzutowanie na typ void.

Obsługa wyjścia – obiekt cout, flagi formatowania

Pamiętamy, że klasa ostream definiuje funkcje operatorową operator<<() dla ściśle określonych typów danych. Dla przykładowej instrukcji:

```
cout << 88;
```

program odszukuje funkcję operatorową o odpowiedniej sygnaturze, w naszym przykładzie , prototyp funkcji operatorowej ma postać:

```
ostream & operator<< (int )
```

Analizując ten prototyp widzimy również, że funkcja zwraca referencje do obiektu klasy ostream. Dzięki temu możliwe jest łączenie danych wyjściowych (sklejanie) co ilustruje następny przykład:

```
…….
int data = 2006;
cout << ”mamy teraz „” << data << „ rok” << endl ;
……..
```

Na ekranie wynikowym pojawi się napis:

mamy teraz 2006 rok

Te przykłady wskazują, że w wielu przypadkach obsługa wejścia/wyjścia w języku C++ z wykorzystaniem obiektu cout jest znacznie prostsza w porównaniu z obsługa wejścia/wyjścia w języku C.

Operatory wstawiania klasy ostream konwertują wartości na postać tekstową. Stosując klasyczne instrukcje, jak np.:

```
int x = 99;
cout  << x ;
```

postać prezentowanej wartości nie zależy od programisty, system stosuje domyślne wartości formatowania.

Obsługa wyjścia – obiekt cout, flagi formatowania

Obowiązują następujące zasady (są drobne różnice pomiędzy starszymi i nowszymi implementacjami języka C++, my opisujemy formatowanie w nowym stylu):

- wartość typu char jest wyświetlana jako znak w polu o szerokości jednego znaku
- łańcuchy są wyświetlane w polu o szerokości równej długości łańcucha
- wartości całkowite (typ int, short, itd.) są wyświetlane jako całkowite liczby dziesiętne, pole ma szerokość taka oby pomieścić wszystkie cyfry i ewentualnie znak minus

Obsługa wyjścia – obiekt cout, flagi formatowania

Wartości typów zmiennoprzecinkowych (float, double, itp.) są wyświetlane zgodnie z następującą konwencją:

- wyświetlamy liczbę w systemie dzietnym, z precyzją 6 cyfr
- zera końcowe są pomijane
- jeżeli po kropce wystąpiłyby same zera to opuszczane są zera i kropka
- jeżeli liczba cyfr przed kropka przekracza sześć, to wypisane zostaną wszystkie , część ułamkowa nie jest pokazana
- liczba jest wyświetlana w notacji zmiennoprzecinkowej lub notacji naukowej. Notacja naukowa jest stosowana gdy wykładnik jest większy od 5 lub mniejszy od -4. W wykładniku, w zależności od implementacji wyświetlane są dwie lub trzy cyfry
- adresy (dodatnie liczby całkowite) wypisywane są w układzie szesnastkowym, z prefiksem ‘0x’
- wartości wskaźnikowe (typ char*) obsługujące napisy wyświetlane są jako znaki, zaczynając od bajtu wskazywanego przez wskaźnik, wyświetlanie kończy się po napotkaniu znaku końca łańcucha
- wartości logiczne (typ bool) wyświetlane są jako liczby całkowite 1 (prawda) i 0 (fałsz)

Obsługa wyjścia – obiekt cout, flagi formatowania

Z każdym strumieniem związany jest zbiór ***flag formatowania***. Flagi formatowania określają sposób wyświetlania informacji, w klasie ios zadeklarowana została wyliczeniowa maska bitowa fmtflags. Mamy 18 dostępnych flag formatowania:

- adjustfield tworzy pole justowania
- basefield ustala pole podstawy systemu liczbowego
- boolalpha wartość logiczna jako true lub false
- dec system dziesiętny
- fixed format ustalony
- floatfield format zmiennoprzecinkowy
- hex system szesnastkowy
- internal wyrównanie obustronne
- left wyrównanie do lewej
- oct system ósemkowy
- right wyrównanie do prawej
- scientific notacja naukowa
- showbase pokaż symbol systemu liczbowego
- showpoint pokaż kropkę dziesiętną
- showpos znak + przed liczbą dodatnią
- skipws zignoruj wiodące białe znaki
- unitbuf opróżnij bufor
- uppercase duże E w zapisie naukowym

Obsługa wyjścia – obiekt cout, flagi formatowania

Zmiany domyślnych ustawień ilustruje kolejny program, stosując wybrane flagi.

Do ustawiania flag formatowania służy funkcja setf(), a do odwoływania zmian służy funkcja unsetf(). Aby ustawić np. ustawić flagę scientific możemy użyć następującej instrukcji:

```
cout.setf(ios::scientific);
```

Użycie nazwy flagi musi być poprzedzone kwalifikatorem ios::, ponieważ flaga jest wartością wyliczeniową zdefiniowana w klasie ios. Flagi możemy łączyć , tak jak w instrukcji:

```
cout.setf(ios::uppercase | ios::scientific);
```

Oby odwołać nowe ustawienia (przywrócić poprzednie) musimy posłużyć się funkcją unseft():

```
cout.unsetf(ios::showpos);
```

Obsługa wyjścia – obiekt cout, flagi formatowania

```
//ustawianie flag
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{ float x = 100;
 bool z = 1;
 cout.setf(ios::showpoint);
 cout.setf(ios::showpos);
 cout << x;                    //na ekranie mamy: +1.000000
 cout.setf(ios::uppercase | ios::scientific);
 cout << "\n" << x;          //na ekranie mamy: +1.000000E+02
 cout.unsetf(ios::showpos);
 cout << "\nzmienna logiczna : " << z;
 cout.setf(ios::boolalpha);
 cout << "\nzmienna logiczna : " << z;
  getche();
  return 0;
}
```

Po uruchomieniu programu otrzymujemy następujący wynik:

```
+100.000
+1.000000E+02
zmienna logiczna : 1
zmienna logiczna : true
```

Obsługa wyjścia – obiekt cout, flagi formatowania

```
//określanie stanu flag
//H.Schildt, Programowanie C++, str. 425
#include <iostream>
#include <conio.h>
using namespace std;
void stan_flag();
int main()
{ stan_flag();                    //flagi domyslne
 cout.setf(ios::right | ios::showpoint | ios:: fixed);
 stan_flag();                     //flagi ustawione
  getche();
  return 0;
}
void stan_flag()
{ ios::fmtflags f;
  long i;
  f = (long) cout.flags();
  for (i = 0x4000; i; i = i >> 1)     //”magiczna liczba” 0x4000
      if (i & f) cout << "1 ";
      else cout << "0 ";
      cout << " \n";
}
```

Zauważmy, że kilka flag możemy połączyć logicznym operatorem OR, tak jak w instrukcji:

cout.setf(ios::right | ios::showpoint | ios:: fixed);

W wyniku wykonania programu mamy następujący wydruk:

0 0 0 0 0 1 0 0 0 0 0 0 0 0 1

0 1 0 0 0 1 0 1 0 0 1 0 0 0 1

Obsługa wyjścia – obiekt cout, flagi formatowania

```
//formatowane wyjście
//ustawianie formatow, flagi
#include <iostream>
#include <conio.h>
#include <math.h>
using namespace std;
void przyklad(int,char*);
int main()
{
 char znak = 'a';
 int x = 123456789;
 double Pi = 3.14159265;
 // 1 - drukuje wartosc domyslnie
 przyklad(1, "tryb domyslny");
 cout << "\n     " << znak ;
 cout << "\n     " << x;
 cout << "\n     " << Pi;
 // 2 - drukuje wartosc double
 przyklad(2, "typ double");
 cout << "\n     " << Pi/1000;
 // 3 - notacja naukowa
 przyklad(3, "notacja naukowa");
 cout << "\n     " << Pi/100000;
 // 4 - kod ASCII dla a
  przyklad(4, "kod ASCII dla a");
  cout << "\n     " << int(znak);
 // 5 - znak dla kodu ASCII 65
  przyklad(5, "znak dla kodu ASCII 65");
  cout << "\n     " << (char)65;
  // 6 - zapis szesnastkowy
  przyklad(6, "zapis szesnastkowy");
  cout << hex << "\n     " << x;
  // 7 - zapis osemkowy
  przyklad(7, "zapis osemkowy");
  cout << oct << "\n     " << x;
  dec(cout);    //odwolywanie zapisu osemkowego
  // 8- ustawianie precyzji
  przyklad(8, "ustawianie precyzji - 10 cyfr");
  cout.precision(10);
  cout << "\n     " << M_PI;
  // 9- ustawianie precyzji
  przyklad(9, "ustawianie precyzji - 2 cyfry");
  cout.precision(2);
  cout << "\n     " << M_PI;
    getche();
    return 0;
}
void przyklad(int n, char* napis)
{   cout << "\n[" << n << "]  "<< napis;
}
```

Obsługa wyjścia – obiekt cout, flagi formatowania

Rezultat działania programu ma postać:

```
[1]  tryb domyślny
a
123456789
3.14159
[2] typ double
0.00314169
[3] notacja naukowa
3.14159e-05
[4] kod ASCII dla a
97
[5] znak dla kodu ASCII 65
A
[6] zapis szesnastkowy
75bcd15
[7] zapis osemkowy
726746425
[8] ustawianie precyzji – 10 cyfr
3.141592654
[9] ustawianie precyzji – 2 cyfry
3.1
```

Obsługa wyjścia – obiekt cout, flagi formatowania

Należy pamiętać, że ***pewne ustawienia (!)*** działają tylko do momentu wyświetlenia pierwszego elementu, inne ustawienia do końca działania programu.

We fragmencie

```
przyklad(7, "zapis osemkowy");
 cout << oct << "\n     " << x;
 dec(cout);    //odwolywanie zapisu osemkowego
 // 8- ustawianie precyzji
 przyklad(8, "ustawianie precyzji - 10 cyfr");
 cout.precision(10);
 cout << "\n     " << M_PI;
```

należy przywrócić tryb zapisu dziesiętnego:

```
dec(cout);    //odwolywanie zapisu osemkowego
```

co jest równoważne zapisowi:

```
cout << dec;
```

Bez przywrócenia flagi domyślnej (zapis w systemie dziesiętnym) tryb ósemkowy nadal by obowiązywał i liczby byłyby wyświetlane w tym systemie.

Obsługa wyjścia – obiekt cout, flagi formatowania

Funkcja składowa width() służy do ustalania szerokości pola w którym mają być wyświetlane dane.

Rozważmy następujące instrukcje:

```
int x = 13;
cout.width(15);
cout << x;
```

Po wykonaniu tego fragmenty, wartość 13 umieszczona zostanie w polu o szerokości 15 znaków i dosunięta do prawego końca tego pola.

Takie justowanie tekstu nosi nazwę „wyrównywania do prawej”. Metoda width() działa tylko na sposób wyświetlenia pierwszego elementu, następny element ma już szerokość domyślną. Dlatego w kolejnym programie ustawianie szerokości pola następuje wewnątrz pętli.

Obsługa wyjścia – obiekt cout, flagi formatowania

```
//ustawianie wypełnienia i szerokosci
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{char *towar[3] = {" telewizor"," radio"," kamera",};
 long cena[3] = {12990, 213, 1988};
 for (int i = 0; i<3; i++)
  { cout.fill('_');
    cout.width(15);
    cout << towar[i] << ": ";
    cout.fill('.');
    cout.width(9);
    cout << cena[i] << " PLN\n";
   }
   getche();
   return 0;
}
```

Rezultatem działania programu jest wydruk:

_____ telewizor: ….12990 PLN
_____ radio: …… 213 PLN
_____ kamera: ….. 1988 PLN

Obsługa wyjścia – obiekt cout, flagi formatowania

Domyślnie obiekt cout wypełnia niewykorzystane pola znakami spacji. Programista może ustalić sposób wypełniania pól przy pomocy metody fill(). W celu wypełnienia pól znakami gwiazdki możemy zastosować instrukcję:

cout.fill('*')

Ustawienie znaku wypełnienia pola obowiązuje do momentu kolejnej zmiany znaku, działa więc inaczej niż ustalenie szerokości pola.

Należy pamiętać, że operator wyprowadzania << nie umieszcza automatycznie znaku nowej linii. Programista musi sam zadecydować o umieszczeniu takiego znaku mając do dyspozycji albo symbol \n lub endl. Wydaje się, że endl jest bardziej użyteczny ponieważ nie tylko wstawia znak nowej linii, ale także czyści bufor wyjściowy. Do ***opróżniania bufora wyjściowego*** służyć może także funkcja flush() lub stosując wyrażenie:

cout << flush;

ponieważ klasa ostream przeciąża operator wstawiania << w taki sposób, że powyższa instrukcja zastępuje wywołanie funkcji flush() jak w poniższym przykładzie:

flush(cout);

Instrukcja flush ***nie wstawia*** znaku końca linii.

Znak nowej linii można umieszczać na kilka sposobów:

```
cout << ”podaj dane: ” << ”\n” ;
cout << ”podaj dane: \n ” ;
cout << ”\n\npodaj dane: ” << ”\n” ;
```

Zadanie 1

Napisz program który wyświetli dane w następujący sposób:

Lola Kwiatek: -----500zl
Buba Zalotna: ----1500zl

programie mamy dwie pracownice i ich nagrody. Należy zastosować flagi do wypełnienia miejsca (flaga fill()) oraz szerokość wydruku (metod width()).

Wynik:

Lola Kwiatek: -----500zl
Buba Zalotna: ----1500zl

Zadanie 2

Ustalenie precyzji wyświetlania liczb zmiennoprzecinkowych i znaku +.

Napisz program, który wyświetli wartość liczby PI z dokładnością standardowa (domyślnie jest to 5 miejsc po przecinku), z dokładnością jednego miejsca po przecinku, z dokładnością 7 miejsc po przecinku oraz ze znakiem.

Wykorzystać metodę setf() oraz funkcję precision().

Wynik:

```
standard,              Pi = 3.14159
precision(2),          Pi = 3.1
precision(8),          Pi = 3.1415926
liczba ze znakiem,     Pi = +3.1415926
```

Wykład 5

KONIEC