Programowanie obiektowe

Paweł Mikołajczak, 2019

7. Operacje wejścia/wyjścia
Part II

Web site: informatyka.umcs.lublin.pl

Wykład 6 *Operacje wejścia/wyjścia*

- ***Wstęp.***
- ***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.***

Obsługa wyjścia – obiekt cout,

Do wyświetlania znaków oraz łańcuchów klasa ostream udostępnia dodatkowo metody put() oraz write().

Metoda put() ma następujący prototyp:

```
ostream & put ( char);
```

Metoda write() posiada następujący prototyp szablonu:

```
typedef charT char_type;
basic_ostream<chart, traits>& write(const char_type* s, streamsize n);
```

argumentami funkcji write() są adres wyświetlanego łańcucha oraz ilość wyświetlanych znaków.

Obsługa wyjścia – obiekt cout,

```
// Metoda put() i write()
#include <iostream>
#include <cstring>     // albo string.h
#include <conio.h>
using namespace std;
int main()
{  const char *miasto = "Lublin";
  const char *napis = " jest ";
  char c1 = 'T';
  char c2 = 'o';
  int d1 = strlen(miasto);
  int d2 = strlen(napis);
  cout << "\nznak c1 = ";
  cout.put(c1);
  cout << "\nnapis   = ";
  cout.write(napis, d1);
  cout << "\n";
  cout.put(c1).put(c2);
  cout.write(napis, d1);
  cout.write(miasto, d2);
  cout << "\n";
  cout.put(65);
  cout.put('a');
  getche();       return 0;   }
```

W wynikiem wykonania tego programu jest komunikat:

```
znak c1  = T
napis      =   jest
To jest Lublin
Aa
```

Obsługa wyjścia – obiekt cout,

Matoda put() (podobnie jak write()) jest składową klasy, wywoływana jest na rzecz obiektu, tak jak w instrukcji:

```
cout.put(c1);
```

Funkcja put() zwraca referencje, wobec czego można łączyć dane wyjściowe:

```
cout.put(c1).put(c2);
```

Argumentem funkcji put() może być typ char oraz typ int (kodem znaku A jest kod ASCII równy 65) :

```
cout.put(65);
cout.put('a');
```

Metoda write() jest funkcją składową wywoływaną na rzecz obiektu cout :

```
cout.write(napis, d1);
```

Do wyznaczenia długości łańcucha wykorzystano funkcję strlen() :

```
int d1 = strlen(miasto);
```

Obsługa wyjścia – manipulatory do formatowania

Stosunkowo niedawno uporządkowano i rozszerzono zestaw specjalnych funkcji zwanych ***manipulatorami*** do zmieniania parametrów formatowania strumienia. Aby uzyskać dostęp do manipulatorów należy do programu włączyć plik nagłówkowy <iomanip>. Zestaw dostępnych manipulatorów pokazany jest w tabeli. Manipulator można traktować jako obiekt wstawiany do strumienia, który modyfikuje działanie obiektu strumienia.

Manipulator	Opis
boolalpha	Literowa prezentacja wartości typu
booldec	Wartość liczbowa w systemie dziesiętnym
endl	Wprowadza znak nowej linii i opróżnia strumień
ends	Wstawia znak końca strumienia
fixe	Wymusza zapis ze stałym znakiem przecinka dziesiętnego
flush	Opróżnia bufor strumienia
hex	Wartość liczbowa w systemie szesnastkowym
internal	Wypełnianie pola na pozycji wewnętrznej
left	Wyrównanie pola do lewej
noboolalpha	wyłącza ustawienie boolalpha

Obsługa wyjścia – manipulatory do formatowania

noshowbase	Wyłącza ustawienie showbase
noshowpoint	Wyłącza ustawienie showpoint
noshowpos	Wyłącza ustawienie showpos
noskipws	Wyłącza ustawienie skipws
nounitbuf	Wyłącza ustawienie unitbuf
nouppercase	Wyłącza ustawienie uppercase
oct	Wartość liczbowa w systemie ósemkowym
resetiosflags((fmtflags f)	Wyłącza flagi wymienione w f
wright	Wyrównuje pola do prawej
scientific	Wymusza wykładniczy zapis liczby

Obsługa wyjścia – manipulatory do formatowania

setbase(int base) Ustawia podstawę do zapisu liczb całkowitych (base może mieć wartości 8, 10 i 16)

setfill(int ch) Ustawia znak wypełnienia

setiosflags(fmtflags f) Ustawia flagi wymienione w f

setprecision(int n) Ustala wartość dokładności

setw(int w) Ustala szerokość pola jako w

showbase Generuje przedrostek określający podstawę systemu liczbowego

showpoint Generuje znak przecinka dziesiętnego w wartościach zmiennoprzecinkowych

showpos Generuje znak + dla nieujemnych wartości

skipws Ustawia pomijanie w strumieniu znaków odstępów

unitbuf Opróżnia bufor po każdej operacji formatowania

uppercase Ustawia stosowanie wielkich liter

ws Opuszcza puste znaki wiodące

Obsługa wyjścia – manipulatory do formatowania

```
//Manipulatory i flagi
#include <iostream>
#include <iomanip>   //plik dla manipulatorow
#include <cstdio>      //plik dla getchar()
using namespace std;

int main()
{
 double x = 12345.55;
//funkcje i flagi ios cout.fill('_');
 cout.width(15);
 cout.setf(ios::showpos);
 cout << x << endl;
//manipulatory
 cout << setfill('_') << setw(15) << showpos << x ;
  getchar();
  return 0;
}
```

W wyniku działania programu otrzymujemy następujący komunikat na ekranie:

_____+12345.5
_____+12345.5

Obsługa wyjścia – manipulatory do formatowania

Aby wyświetlić liczbę x bardziej dokładniej, musimy ustawić precyzję. Można to zrobić wykorzystując manipulator setprecision(), zmodyfikowana instrukcja ma postać:

```
cout << setfill('_') << setw(15) << showpos << setprecision(8)<<x ;
```

Po wykonaniu takiej zmiany w naszym programie wydruk na ekranie ma postać:

_____+12345.5

_____+12345.55

Większość manipulatorów ustawia pojedyncze znaczniki formatu. Przedstawicielem takich manipulatorów jest użyty w programie manipulator showpos służący do wyświetlenia znaku + (plus). Są też manipulatory wykonujące bardziej złożone operacje. Przykładem takiego manipulatora jest manipulator endl, który wstawia do strumienia znak nowego wiersza i opróżnia bufor strumienia. Wyrażenie:

```
cout << endl;
```

jest równoważne następującym:

```
cout << '\n' ;
cout.flush();
```

Obsługa wyjścia – manipulatory do formatowania

Niektóre manipulatory przyjmują argumenty. Takimi manipulatorami są np. pokazany w programie manipulator setw(15). Manipulator setw() ustawia szerokość pola , w naszym przypadku jest to szerokość 15 znaków. Wyrażenie:

```
cout << setw(15);
```

jest równoważne wyrażeniu:

```
cout.width(15);
```

Innym przykładem jest użyty manipulator setfil('_') , przy pomocy którego definiujemy znak wypełnienia pól. Gdy zachodzi potrzeba wstawienia np. gwiazdek do wypełnienia, możemy zastosować wyrażenie setfill('*').

Podobne działanie jak funkcja składowa setf() ma manipulator setiosflags().

Obsługa wyjścia – manipulatory do formatowania

```
// Manipulatory setiosflags() i boolalpha
#include <iostream>
#include <iomanip>   //plik dla manipulatorow
#include <cstdio>      //plik dla getchar()
using namespace std;
int main()
{double x = 12345.5;
 int y = 16;
 bool z = 1;
 cout << setiosflags(ios::showpos);
 cout << x << endl;
 cout << setiosflags(ios::showbase);
 cout <<"liczba dziesietna " << y << " w systemie oktalnym to: "<< oct << y ;
 cout << endl << dec << "wartosc logiczna z to " << z << endl;
cout << "wartosc logiczna " << noshowpos << z << boolalpha << " to " << z;
  getchar();
  return 0;
}
```

W wyniku działania tego programu mamy następujący komunikat na ekranie:

```
+12345.5
liczba dziesietna +16 w systemie oktalnym to: 020
wartosc logiczna z to +1
wartosc logiczna 1 to true
```

Obsługa wyjścia – manipulatory do formatowania

Manipulator setiosflags() ustawił w naszym programie flagi showbase i showpos.

Manipulator oct i dec ustawiały system liczbowy (w tym przypadku ósemkowy i dziesiętny) w jakim wyświetlane były wartości zmiennej y i z. Manipulator boolalpha umożliwia wprowadzanie i wyprowadzanie wartości logicznych przy użyciu słów „true” i „false” jako zamiennik dla wartości logicznych 1 i 0.

Obsługa wejścia – obiekt cin, flagi formatowania

W bibliotece standardowej do obsługi strumienia wejściowego najczęściej używany jest obiekt cin łącznie z operatorem >>, nazywanym także ***operatorem pobierania***.

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 wejścia – obiekt cin, flagi formatowania

Wejście realizowane jest za pośrednictwem przeciążonego operatora przesunięcia bitowego w prawo operator>>(). Operator pobierania nazywany jest także ekstraktorem (ang. extractor). Ekstraktor odczytuje dane ze strumienia wejściowego i umieszcza je w obiekcie docelowym, jak poniżej:

```
cin >> x >> y ;
```

W powyższym przykładzie zmienne x i y zostaną wypełnione wartościami zgodnie z typem naszych zmiennych. Należy pamiętać, że na wyjściu możemy ściśle kontrolować format *wyprowadzanych* danych (strumien wyjściowy). Podczas wyodrębniania danych *wejściowych* (strumień wejściowy) zazwyczaj nie ma tak rygorystycznej kontroli formatu. W takiej sytuacji podczas wczytywania np. danych zmiennoprzecinkowych musimy być przygotowania na różne formaty. Bardzo często nie wiemy czy wprowadzane dane będą w formacie wykładniczym czy dziesiętnym. Nie wiemy z góry czy liczba dodatnia będzie poprzedzona znakiem plus czy nie. Ekstraktor powinien być przygotowany do odczytu wartości w dowolnym dozwolonym formacie dla oczekiwanego typu wartości. Ekstraktor powinien odczytać np. wartość 1.5 która może być wprowadzona w jednym z możliwych formatów:

```
+1.5E00 +1.51.500.15E+1
```

Obsługa wejścia – obiekt cin, flagi formatowania

Obiektu cin używamy najczęściej w następujący sposób:

```
cin >> miejsce_przechowywania_wartości
```

Parametr *miejsce_przechowywania_wartości* może być nazwą zmiennej, referencją, wyłuskanym wskaźnikiem a także składową struktury lub składową klasy. Kolejny przykład pokazuje proste sposoby wprowadzania danych, pokazuje też pewne pułapki. Dane można wprowadzać pojedynczo:

```
cout << "wprowadz int x : ";
cin >> x ;
```

a także przy użyciu wielu operatorów >>, tak jak w przykładzie:

```
cout << " \npodaj nowe wartosci x,y oraz b1:\n";
cin >> x >> y >> b1;
```

można użyć manipulatora:

```
cout << "wprowadz logiczne b2 (false lub true): ";
 cin >> boolalpha >> b2;
```

Obsługa wejścia – obiekt cin, flagi formatowania

```
// wprowadzanie danych, obiekt cin
#include <iostream>
#include <iomanip>
#include <conio.h>
using namespace std;
int main()
{ int x;
 double y;
 bool b1, b2;
 cout << "wprowadz int x : ";
 cin >> x ;
 cout << "wprowadz double y: ";
 cin >> y ;
 cout << "wprowadz logiczne b1 (0 lub 1): ";
 cin >> b1;
 cout << "wprowadz logiczne b2 (false lub true): ";
 cin >> boolalpha >> b2;
 cout << x <<"  "<< y <<"  "<< b1 <<"  "<< boolalpha << b2;
 cout << " \npodaj nowe wartosci x,y oraz b1:\n";
 cin >> x >> y >> b1;
 cout << "x = " << x << " y = " << y << " b1 = " << b1;
  getche();
  return 0;
}
```

Oto rezultat wykonania tego programu:

```
wprowadz int x : 22
wprowadz double y : 2.2
wprowadz logiczne b1 (0 lub 1): 0
wprowadz logiczne b2 (false lub true): true
22 2.2 0 true
podaj nowe wartości x,y oraz b1:
44 4.4 true
x = 44  y = 4.4  b1 = true
```

Obsługa wejścia – obiekt cin, flagi formatowania

Należy pamiętać o manipulatorze boolalpha. Gdyby ponowne wprowadzanie danych miało postać:

```
podaj nowe wartości x,y oraz b1:
44 4.4 1
```

To w wyniku otrzymamy fałszywy wydruk:

```
x = 44  y = 4.4  b1 = false
```

i to bez ***żadnego ostrzeżenia.***

Problem leży w następującym fragmencie programu:

```
cout << "wprowadz logiczne b2 (false lub true): ";
cin >> boolalpha >> b2;
```

Stan strumienia jest tak ustawiony, że żąda wprowadzania danych boolowskich tylko jako true lub false a nie jako 0 lub 1.

obiekt cin, stan strumienia

Na szczęście obiekty cin i cout zawierają daną składową opisującą stan strumienia co pozwala na obsługę sytuacji gdy pojawiają się nieprawidłowe dane wejściowe. Jest to bardzo obszerne i skomplikowane zagadnienie.

W trybach jednoznakowych operator >> odczytuje dany znak i zapisuje go we wskazanym miejscu.

Jeżeli wprowadzamy dane wieloznakowe, operator odczytuje wszystkie znaki zgodne z typem docelowym.

W kolejnym programie musimy wprowadzić dwie liczby całkowite (typ int).

obiekt cin, stan strumienia

```
//wprowadzanie danych, obiekt cin, kontrola błędów
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{  int x, y;
   cout << "wprowadz int x i y : ";
   cin >> x >> y;
   cout << "x + y = " << x + y << endl;
    getche();
    return 0;
 }
```

Poprawne dane:
wprowadz int x i y : 20 20
x + y = 40

Niepoprawne dane:
wprowadz int x i y : 20 2o
x + y = 22

Wprowadzono literę o

obiekt cin, stan strumienia

Bez ***żadnego ostrzeżenia*** program akceptuje błędne dane. W ostatnim przykładzie operator natrafił na znak ”o” , wobec czego ostatnim przeczytanym znakiem jest cyfra 2 i taka została dodana do poprzedniej 20. Znak o pozostał w buforze i w kolejnej instrukcji ***cin*** od niego rozpocznie się odczyt. Możemy kontrolować takie nieoczekiwane błędy korzystając z faktu, że w momencie natrafienia na złą daną operator pobierania zwróci wartość 0 (false). Taką prostą obsługę błędów wejścia pokazujemy w kolejnym programie.

obiekt cin, stan strumienia

```
// obiekt cin, kontrola błędów
#include <iostream>
#include <conio.h>
using namespace std;

int main()
{ int liczba, suma = 0;
  cout << "wprowadz dane : ";
  while (cin >> liczba)
       suma += liczba;
  cout << "ostatnia liczba  : " << liczba << endl;
  cout << "suma = " << suma<< endl;
    getche();
    return 0;
}
```

```
wprowadz dane : 10 20 30 40 w
ostatnia liczba  : 4199019
suma = 100
```

System „zgubił się”, ale suma jest prawidłowa!

obiekt cin, stan strumienia

W momencie wprowadzenia znaku „w” pętla *while* przerwała działania a program podał prawidłowy wynik. Niestety nie została poprawnie odczytana wprowadzona ostatnia liczba. Niezgodność danych wejściowych z oczekiwanym formatem powoduje, że wyrażenie

cin >> liczba

zwraca wartość *false* i oczywiście wtedy przerwane jest wykonywanie pętli *while*.

Aby program opisany program mógł działać bardziej elegancko, ***potrzebujemy bardziej wyrafinowanych narzędzi***. Bardzo pomocne są informacje o stanie strumieni wejścia i wyjścia.

obiekt cin, stan strumienia

Aby opisać stan strumieni, określono cztery stałe typu *iostate*, które są znacznikami. Typ *iostate* stanowi składową klasy *ios_base*. Tabela pokazuje te stałe. Każda taka stała jest pojedynczym bitem i może przyjmować wartość 1 (*ustawiony*) lub 0 (*skasowany*).

Stała	Opis
goodbit	Ustawiony, gdy operacja jest wykona poprawnie, nie jest ustawiony żaden inny bit
eofbit	Ustawiony gdy został napotkany znak końca pliku
failbit	Ustawiony gdy operacja wejścia/wyjścia zakończyła się niepowodzeniem (nieudany odczyt oczekiwanych znaków)
badbit	Ustawiony gdy wystąpił błąd krytyczny, uszkodzony strumień

obiekt cin, stan strumienia

Gdy zostanie napotkany koniec pliku operacja *cin* ustawia bit *eofbit*. Gdy odczyt znaku kończy się niepowodzeniem zostaje ustawiony bit *failbit*. Element *badbit* jest ustawiony, gdy strumień został uszkodzony wskutek nieznanego błędu. Widzimy, że gdy wszystkie trzy bity są wyzerowane, oznacza to , że wszystko jest w porządku, odczyt jest poprawny. Wartość *goodbit* została zdefiniowana jako wartość 0. Dlatego też jej ustawienie oznacza, że wszystkie inne bity są wyczyszczone. Jest to trochę mylące, ponieważ nie oznacza ustawienia jednego z bitów ale wyczyszczenie wszystkich bitów. Znaczniki pokazane w tabeli są obsługiwane przez klasę *basic_ios* i dlatego są obecne we wszystkich obiektach typu *basic_istream* oraz *basic_ostream*.

obiekt cin, stan strumienia

W klasie *ios_base* znajdują się **metody** do kontroli i modyfikacji stanu strumieni. Najważniejsze z nich pokazane są w tabeli.

Metoda	Opis
good()	Zwraca wartość true, gdy strumień jest poprawny (ustawiony jest znacznik goodbit)
eof()	Zwraca wartość true, gdy został napotkany znak końca pliku (ustawiony znacznik eofbit)
fail()	Zwraca wartość true, gdy wystąpił błąd (ustawiony znacznik failbit lub badbit)
bad()	Zwraca wartość true, gdy wystąpił błąd krytyczny (ustawiony znacznik badbit)
rdstate()	Zwraca bieżący zestaw znaczników
clear()	Czyści wszystkie znaczniki
clear(state)	Czyści wszystkie znaczniki i nadaje im wartość określoną przez argument state
setstate(state)	Ustawia dodatkowe znaczniki określone przez argument state

obiekt cin, stan strumienia

W języku C++ mamy możliwość obsługi błędów zgłaszanych przez znaczniki stanu strumienia – można do tego celu wykorzystać obsługę ***wyjątków***.

W tabeli pokazane są funkcje składowe strumieni, służące do obsługi wyjątków.

Metoda	Opis
exceptions(flags)	Ustawia znaczniki, generuje wyjątki
exceptions()	Zwraca znaczniki, które powodują generowanie wyjątków

Wywołanie funkcji exceptions() bez argumentów powoduje zwrócenie aktualnego zestawu znaczników, dla których obsługiwane są wyjątki. Jeżeli funkcja zwróci wartość goodbit, wyjątki nie są generowane.

obiekt cin, stan strumienia

Gdy chcemy aby wyjątki były generowane dla wszystkich znaczników możemy użyć instrukcji:

```
strm.exceptions(std::ios::eofbit | std::ios::failbit | std::ios:: badbit);
```

wyjątki nie będą generowane, gdy użyjemy instrukcji:

```
strm.exceptions(std:: ios::goodbit);
```

Klasa wyjątków ios_base::failure wywodzi się z klasy std::exception i dlatego posiada metodę what().

Jedyną metodą pozwalającą na pobranie informacji o błędzie jest komunikat błędu zwrócony przez funkcje what().

obiekt cin, stan strumienia, what()

```
// wprowadzanie danych, obiekt cin, stan strumienia
#include <iostream>
#include <exception>
#include <conio.h>
using namespace std;
int main()
{ cin.exceptions(ios_base::failbit);
  int liczba, suma = 0;
  cout << "wprowadz dane : ";
  try { while (cin >> liczba)
              suma += liczba;
      } catch(ios_base::failure & er)
         { cout << er.what() << endl;
         }
  cout << "ostatnia liczba  : " << liczba << endl;
  cout << "suma = " << suma<< endl;
    getche();
    return 0;
}
```

W wyniku mamy następujący wydruk:

```
wprowadz dane : 12 20 30 40 w
ios failure
ostatnia liczba   : 40
suma w 100
```

obiekt cin, stan strumienia

Otrzymaliśmy eleganckie rozwiązanie naszego problemy. Powyższy program został przerwany ponieważ został uszkodzony strumień. Jeżeli chcemy kontynuować nasz program musimy przywrócić dobry stan strumienia. Jak wiemy możemy do tego celu użyć metody clear(). Metoda clear() wyzeruje znaczniki, ale niepoprawne dane wejściowe, które przerwały wykonywanie programu dalej czekają w kolejce na wczytanie. *Należy te dane pominąć*. Jednym z rozwiązań jest zastosowanie funkcji isspace(), która zwraca wartość true, gdy jej argumentem jest znak odstępu (spacja). Pętla while pozwala pominąć złe dane:

```
while (!isspace(cin.get()))
        continue;
```

Możemy również pominąć resztę całego wiersza a nie tylko następne słowo gdy użyjemy następującej instrukcji:

```
while(cin.get() != '\n')
        continue;
```

obiekt cin, stan strumienia, clear()

```
// obiekt cin, stan strumienia, metoda clear()
#include <iostream>
#include <exception>
#include <conio.h>
using namespace std;
int main()
{ cin.exceptions(ios_base::failbit);
  int liczba, suma = 0;
  cout << "wprowadz dane : ";
  try { while (cin >> liczba)
              suma += liczba;
      } catch(ios_base::failure & er)
         { cout << er.what() << endl;
         }
  cout << "ostatnia liczba  : " << liczba << endl;
  cout << "suma = " << suma<< endl;
  cin.clear();
  while (!isspace(cin.get()))
         continue;
  int n;
  cout << "podaj liczbe : ";
  cin >> n;
  cout << " n* suma = " << n*suma<< endl;
  getche();      return 0;    }
```

Po uruchomieniu tego programu mamy wynik:

```
wprowadz dane : 10 20 30 w
ios failure
ostatnia liczba  : 30
suma = 60
podaj liczbe : 10
 n* suma = 600
```

obiekt cin, stan strumienia, fail()

```
// obiekt cin, stan strumienia, metoda fail()
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{ int liczba, suma = 0;
  cout << "wprowadz dane : ";
   while (cin >> liczba)
              suma += liczba;
  cout << "suma = " << suma << endl;
  if (cin.fail() && !cin.eof() )
     {  cin.clear();
        while (!isspace(cin.get()))
          continue;
     }
  else
     { cout << "koniec pracy" << endl;
       exit(1);
     }
  int n;
  cout << "podaj liczbe : ";
  cin >> n;
  cout << " n* suma = " << n*suma<< endl;
  getche();     return 0;  }
```

Niekoniecznie musimy stosować dość skomplikowaną technikę obsługi wyjątków. Możemy bezpośrednio wykorzystać funkcje składowe takie jak np. fail().

Wynik działania programu ma postać:

```
wprowadz dane : 10 20 30 w
suma = 60
podaj liczbe : 3
 n* suma = 180
```

obiekt cin, łańcuchy w stylu C

Wiemy, że istnieje szczególny związek pomiędzy tablicami a wskaźnikami. Nazwa tablicy to adres jej pierwszego elementu. W języku C napis traktowany jest jako tablica znaków. Rozważmy następującą instrukcję:

```
char nazwisko[30] = ”Kowalski” ;
cout << ”Pan ” << nazwisko ;
```

W C++ mamy możliwość traktowania łańcuchów w stylu języka C.

Nazwa tablicy jest adresem pierwszego elementu tej tablicy.

W naszym przykładzie zmienna tablicowa *nazwisko* dla obiektu *cout* jest adresem litery K, od której zaczyna się cały napis, ostatnim znakiem łańcucha jest znak NULL (0). Jeżeli obiekt *cout* otrzyma adres pierwszego znaku, zostanie wyświetlony cały napis. Widzimy, że możemy użyć wskaźnika na daną *char* jako parametr dla obiektu *cout*. W naszym przykładzie nie następuje wysłanie do obiektu *cout* całego napisu a jedynie wysyłany jest adres tego napisu. Napis pochodzący ze strumienia można przypisać tablicy za pomocą *cin.*

obiekt cin, łańcuchy w stylu C

```
// obiekt cin, napisy w konwencji C
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{ char imie[80],nazwisko[80],miasto[80];
  cout << "podaj imie : ";
  cin >> imie;
  cout << "podaj nazwisko : ";
  cin >> nazwisko;
  cout << "podaj miasto : ";
  cin >> miasto;
  cout << imie << "  " << nazwisko << " i  jego miasto " << miasto;
    getche();
    return 0;
 }
```

Działanie programu jest następujące:
podaj imie : Jan
podaj nazwisko : Kowalski
podaj miasto : Lublin
Jan Kowalski i jego miasto

obiekt cin, łańcuchy w stylu C

```
// obiekt cin, napisy w konwencji C
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{ char nazwisko[80],miasto[80];
  cout << "podaj nazwisko : ";
  cin >> nazwisko;
  cout << "podaj miasto : ";
  cin >> miasto;
  cout << nazwisko << " i  jego miasto " << miasto;
    getche();
    return 0;
 }
```

Działanie tego programu ma postać:
podaj nazwisko : Jan Kowalski
podaj miasto: Jan i jego miasto Kowalski

obiekt cin, łańcuchy w stylu C

Wprowadzenia napisu „Jan Kowalski” ***spowodowała nieoczekiwany skutek.*** Należy pamiętać, że cin wczytuje znaki z klawiatury do momentu napotkania pierwszego odstępu (białej spacji). Przy pomocy obiektu cin jesteśmy w stanie odczytać *tylko jeden wyraz*, został on pobrany i wysłany do tablicy *nazwisko*. Oznacza to, że w buforze pozostał drugi wyraz „Kowalski” i on został odczytany przez obiekt cin w drugim wywołaniu. Pewnym usprawnieniem zapobiegającym nieuprawnionemu wczytywaniu kolejnego wyrazu w napisie jest użycie funkcji ignore(). Funkcja ta kasuje zawartość bufora.

Instrukcja

```
cin.ignore(100, '\n');
```

będzie odrzucać 80 kolejno wczytanych znaków, chyba, że wcześniej napotka znak nowego wiersza.

łańcuchy w stylu C , ignore()

```
// obiekt cin, napisy w konwencji C, metoda ignore()
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{ char nazwisko[80],miasto[80];
  cout << "podaj nazwisko : ";
  cin >> nazwisko;
  cin.ignore(100, '\n');   //kasowanie bufora - metoda ignore()
  cout << "podaj miasto : ";
  cin >> miasto;
  cout << nazwisko << " i  jego miasto " << miasto;
    getche();
    return 0;
 }
```

Program da następujący wynik:
podaj nazwisko : Jan Kowalski
podaj miasto : Lublin
Jan i jego miasto Lublin

łańcuchy w stylu C, getline()

```
// obiekt cin, napisy w konwencji C, metoda getline()
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{ char nazwisko[80],miasto[80];
  cout << "podaj nazwisko : ";
  cin.getline(nazwisko, 80) ;
  cout << "podaj miasto : ";
  cin.getline(miasto, 80);
  cout << nazwisko << " i  jego miasto " << miasto;
    getche();
    return 0;
}
```

usprawnienie polega na użyciu metody getline() dzięki której będzie możliwe wczytanie całego wiersza.

Działanie programu ma postać:
podaj nazwisko : Jan Maria Kowalski
podaj miasto : Lublin
Jan Maria Kowalski i jego miasto Lublin

obiekt cin, łańcuchy w stylu C

Dzięki funkcji getline() możemy wczytywać całe wiersze, końcem wiersza jest znak nowego wiersza (‘\n’), który jest generowany przez naciśnięcie klawisza *Enter*.

Funkcja getline() ma dwa parametry:

- pierwszy to nazwa tablicy przechowującej wczytywany napis,
- drugi to maksymalna liczba wczytywanych znaków.

W naszym przypadku wpisaliśmy liczbę 80 co oznacza, że wczytanych będzie 79 znaków, ponieważ ostatnim elementem w naszej tablicy będzie wprowadzany automatycznie znak końca napisu (Null).

obiekt cin, łańcuchy w stylu C

Biblioteka standardowa C++ dostarcza nam kilku użytecznych funkcji składowych służących do odczytywania sekwencji znaków.

W tablicy s oznacza łańcuch znaków, n – liczba znaków, t – ogranicznik

Metoda	Opis
get()	Odczytuje następny znak, zwraca odczytany znak lub EOF
get(c)	Przypisuje kolejny znak do przekazanego argumentu c
get(s, n)	Odczytuje n-1 znaków z sekwencji wskazywanej przez wartość s, kończy odczyt gdy natrafi na znak '\n'
get (s,n,t)	Odczytuje n-1 znaków z sekwencji wskazywanej przez wartość s, kończy odczyt gdy natrafi na znak opisany przez t
getline(s,n)	Odczytuje n-1 znaków z sekwencji wskazywanej przez wartość s, kończy odczyt gdy natrafi na znak '\n', ulepszona
getline(s,n,t)	Odczytuje n-1 znaków z sekwencji wskazywanej przez wartość s, kończy odczyt gdy natrafi na znak opisany przez t
read(s,n)	Odczytuje n znaków w łańcuchu s, kończy gdy natrafi na znak końca pliku
readsome(s,n)	Odczytuje n znaków w łańcuchu s, kończy gdy natrafi na znak końca pliku, zwraca liczbę odczytanych znaków

obiekt cin, łańcuchy w stylu C

Mamy także inne użyteczne funkcje obsługujące znaki. Te funkcje składowe pokazane są w kolejnej tabeli.

Metoda	Opis
gcount()	Zwraca liczbę odczytanych znaków
ignore()	Pobiera i odrzuca znak, ignoruje jeden znak
ignore(n)	Pobiera i odrzuca znaki, ignoruje n znaków
ignore(n, t)	Pobiera i odrzuca znaki, ignoruje n znaków aż do napotkania znaku t
peek()	Zwraca pozycje kolejnego znaku ze strumienia bez jego pobierania.
unget()	Umieszcza ostatni znak ponownie w strumieniu
putback(char c)	Umieszcza ostatni znak ponownie w strumieniu, sprawdza czy c był ostatnim odczytanym znakiem

obiekt cin, łańcuchy w stylu C

Należy zawsze zabezpieczyć program przed otrzymywaniem z zewnątrz (np. klawiatury) niepoprawnych danych. Sam strumień cin prowadzi wstępną weryfikację danych, ponieważ odczytuje jedynie wartości odpowiadające typowi zmiennej, której mają zostać przypisane. Bardzo często zachodzi jednak przypadek, że cin nie odczyta danych a mimo to program jest kontynuowany, tak jakby nie było błędu. Inny problem polega na tym, że gdy program ma instrukcję wprowadzania danych to będzie czekał aż dane nie zostaną wprowadzone (możemy naciskać klawisz *Enter* dziesiątki razy). Te fakty skłaniają nas do stosowania odpowiednich narzędzi służących kontroli stanu strumienia. Obiekt cin ma odpowiednie funkcje składowe, które informują o poprawności wykonania operacji wczytywania danych. Zazwyczaj funkcje te zwracają wartość true lub false. W tabeli pokazano często używane funkcje do sprawdzania stanu strumienia.

obiekt cin, stan strumienia

Metoda	Opis
bad()	Zwraca wartość true , jeśli cin nie wczytał danych
eof()	Zwraca wartość true gdy napotkano koniec pliku lub wejścia
fail()	Zwraca wartość true jeśli cin nie wczytał poprawnie danych
good()	Zwraca wartość true jeśli żadna z funkcji bad(), eof() lub fail() nie zwraca wartości true

łańcuchy w stylu C , good()

```
//obiekt cin, napisy w konwencji C, metoda good()
#include <iostream>
#include <conio.h>
using namespace std;

int main()
{ int rok;
  cout << "podaj rok urodzenia : ";
  cin >> rok;
  if (cin.good())
     cout << "masz "<< 2008 - rok << " lat" << endl;
  else
     cout << "zle dane  - koniec ";
  getche();
  return 0;
}
```

Po uruchomieniu mamy następujący wynik
podaj rok urodzenia : 1947
masz 61 lat

Wynikiem działania programu podczas wprowadzenia znaku o zamiast cyfry 0 może być wynik:

podaj rok urodzenia : 199o
masz 1809 lat

obiekt cin, łańcuchy w stylu C

Prosta pomyłka nie powoduje wyświetlenia żadnego komunikatu, wyniki obliczenia są ***błędne***. Dzieje się tak, ponieważ system odczytuje poprawnie trzy znaki 1, 9 , 9 (które reprezentują liczby całkowite) a odrzuca znak „o”. System zasygnalizuje błąd, gdy pierwsza wprowadzona wartość nie będzie cyfrą:

podaj rok urodzenia : I947

zle dane - koniec

W tym przykładzie podczas wprowadzania danych z klawiatury został wprowadzony znak „I” zamiast cyfry 1.

Widzimy, że pewne błędy są trudne do wychwycenia, na szczęście mamy metody pozwalające radzić sobie z tym problemem.

obiekt cin, łańcuchy w stylu C

W przykładowym programie zamiast instrukcji:

```
if (cin.good());
```

możemy użyć instrukcji:

```
if (cin);
```

lub zamiast

```
cin >> rok;
if (cin.good())
```

możemy napisać wprost:

```
if (cin >> rok)
```

obiekt cin, łańcuchy w stylu C

Gdy wystąpi błąd wykryty w strumieniu, cin ma wartość *false*, jeśli stosowaliśmy funkcje cin.fail() lub cin.bad() zwrócą one wartość *true*. Aby przywrócić strumień cin do stanu początkowego możemy wykorzystać funkcję clear(), która zmienia wskaźnik błędu oraz użyć funkcji ignore() do opróżnienie bufora. Rekomendowana sekwencja instrukcji może mieć postać:

```
while ( ! ( cin >> rok ) )
    {       cin.clear ( );
            cin.ignore ( 80, '\n' );
            cout << "podaj rok urodzenia : " ;
    }
```

Zastosowanie pętli while zapewnia, że fragment programu z instrukcjami wczytywania danych będzie się powtarzał tak długo, dopóki nie zostaną wprowadzone poprawne dane. Niestety gdy wprowadzona zostanie sekwencja danych zawierająca znak na drugim i dalszym miejscu (np. 19o7) błąd nie będzie wykryty.

łańcuchy w stylu C ,wskaźniki

Stwierdzono już poprzednio, że wskazana jest szczególna ostrożność w przypadku obsługi napisów w stylu C przy pomocy wskaźników.

```
//obiekt cin, napisy w konwencji C, wskaźniki
#include <iostream>
#include <conio.h>
using namespace std;

int main()
{ char *nazwisko;
  cout << "podaj nazwisko : ";
  cin >> nazwisko;
  cout << "pan " << nazwisko ;
  cout << " i  jego kolega Zielinski" << endl;
    getche();
    return 0;
 }
```

Po uruchomieniu mamy wynik:
podaj nazwisko : Kowalski
pan Kowalski i jego kolega Zieliński

łańcuchy w stylu C , wskaźniki

```
//obiekt cin, napisy w konwencji C, wskaźniki
#include <iostream>
#include <conio.h>
using namespace std;

int main()
{ char *nazwisko;
  cout << "podaj nazwisko : ";
  cin >> nazwisko;
  cout << "  pan " << nazwisko ;
  cout << "\npodaj nazwisko : ";
  cin >> nazwisko;
  cout << "  i  jego kolega " << nazwisko << endl;
    getche();
    return 0;
 }
```

Otrzymamy wynik:

```
podaj nazwisko : Kowalski
     pan Kowalski
podaj nazwisko : Zieliński
      i jego kolega Zieliński
```

łańcuchy w stylu C , wskaźniki

Próba deklaracji ***dwóch wskaźników typu char nie inicjalizowanych*** w większości kompilatorów kończy się katastrofą. Próba wykonania poniższego programu (po usunięciu komentarza) kończy się awaryjnym zawieszeniem

```
//napisy w konwencji C, wskaźniki – potencjalne problemy
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{   char *nazwisko_1;
  //char *nazwisko_2;    //NIGDY TEGO NIE ROBIC !!
    cout << "podaj nazwisko_1 : ";
    cin >> nazwisko_1;
 // cin >> nazwisko_2;   //NIGDY TEGO NIE ROBIC !!
    cout << "  pan " << nazwisko_1 ;
      getche();
      return 0;
}
```

łańcuchy w stylu C , wskaźniki

Aby wczytać łańcuch do programu, należy przydzielić mu miejsce w pamięci, a następnie pobrać go za pomocą funkcji wejścia. Najczęściej deklaracja łańcucha w stylu C ma postać:

```
char nazwisko [80];
```

Możemy deklarować tablice z inicjalizacją:

```
char nazwisko [80] = ”Kowalski”;
```

Nazwa tablicy to adres jej pierwszego elementu. W naszym przykładzie *nazwisko* jest adresem elementu char zawierającego literę K. Należy pamiętać, że w instrukcji:

```
cout << "  pan " << nazwisko_1 ;
```

napis ” pan ” także jest adresem łańcucha, czyli wskaźnikiem do pierwszego znaku tego napisu. W języku C/C++ łańcuchy zapisywane jako tablice, napisy w cudzysłowach i wskaźniki na znaki są obsługiwane tak samo. Łańcuch możemy deklarować także przy pomocy następującej deklaracji:

```
char * nazwisko = ”Kowalski”;
```

W tym przykładzie, ”Kowalski” jest adresem łańcucha, pokazana instrukcja przypisuje wskaźnikowi *nazwisko* adres łańcucha ”Kowalski”. Literały łańcuchowe są stałymi, wobec tego można napisy deklarować następująco:

```
const char * nazwisko = ”Kowalski”;
```

łańcuchy w stylu C , wskaźniki

Tworzenie niezainicjalizowanego wskaźnika do typu char nie jest polecane, a użycie dwóch takich deklaracji doprowadzi do problemów. Dzieje się tak dlatego, że kompilatory zazwyczaj traktują literały łańcuchowe jako stałe tylko do odczytu, przy próbie ich nadpisania zgłaszają błąd. Co ciekawsze, zazwyczaj kompilatory stosują tylko jedną kopię literału, aby obsłużyć wszystkie jego wystąpienia w programie. Oznacza to, że nie wszystkie literały będą miały zarezerwowane miejsce w pamięci. Gdy mamy już zainicjalizowany jakiś łańcuch, poważny błąd może wystąpić w momencie, gdy będziemy chcieli wczytać łańcuch w miejsce wskazywane przez inny niezainicjalizowany wskaźnik typu char.

```
char *nazwisko_1;
/ / char *nazwisko_2;    //NIGDY TEGO NIE ROBIC !!
cin >> nazwisko_1;
 cin >> nazwisko_2;
```

Może się zdarzyć, że nadpisane zostaną dane już wcześniej istniejące w pamięci. Rekomenduje się przy obsłudze napisów w stylu C używanie dużej tablicy char na dane wejściowe i nie używać do wczytywania danych stałych ani niezainicjalizowanych wskaźników. Podczas wczytywania łańcuchów znakowych zawsze powinno się używać adresów wcześniej zaalokowanej pamięci.

łańcuchy w stylu C , wskaźniki

Poniższy program uruchomi się bez kłopotu, ponieważ użyliśmy zainicjalizowanych wskaźników.

```
// napisy w konwencji C, zainicjalizowane wskaźniki
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{ char *nazwisko_1 = "Kowalski";
  char *nazwisko_2 = "Zielinski";
  cout << "  pan " << nazwisko_1 ;
  cout << " i jego kolega " << nazwisko_2 << endl;
    getche();
    return 0;
 }
```

W języku C++ obecnie rekomenduje się do obsługi napisów wykorzystywać klasę **string** i jej metody.

Wejście/wyjście

W zasadzie w każdym omawianym programie języka C++ mamy do czynienia z operacjami wejścia /wyjścia. Korzystamy z obiektów ***cin*** oraz ***cout***. Te zagadnienia są omawiane w pierwszej kolejności. Z drugiej strony, należy wiedzieć, że obsługa wejścia /wyjścia jest w języku C++ jest realizowana przy wykorzystaniu bardzo skomplikowanych technik – korzystamy z klas, klas pochodnych, przeciążanych funkcji, funkcji wirtualnych, szablonów i wielodziedziczenia. Podstawową obsługą wejścia/wyjścia zajmują się obiekty ***cin*** klasy ***istream*** oraz ***cout*** klasy ***ostream*** a obsługa plików obiekty klas ***ifstream*** oraz ***ofstream***.

W organizacji procesów wejścia/wyjścia kluczowe znaczenie ma pojęcie twz. ***strumienia bajtów*** (analogia : ruch kropel wody w strumieniu). Na wejściu program pobiera bajty strumienia wejściowego z klawiatury, dysku czy innego programu. Na wyjściu bajty strumienia wyjściowego wysyłane są na ekran , pamięci masowej do drukarki czy do innego programu. Dzięki takiej koncepcji, język C++ analizuje jedynie strumień danych, nie musi wiedzieć skąd one pochodzą. Dzięki takiej koncepcji klawiaturę, ekran, dysk twardy możemy traktować jako pliki (zbiór bajtów).

Wejście/wyjście

W klasie ***istream*** zadeklarowana jest funkcja operatorowa ***operator >> ().*** Ten operator służy do wprowadzania danych ze strumienia ***cin,*** standardowo podpięty jest do klawiatury.

Przeciążony operator **<<** jest skojarzony z buforowanym strumieniem **cout** i używany jest standardowo do wyprowadzania danych na ekran (można go przekierować na inne urządzenie).

W najczęściej wykorzystywanych klasach ***ios, istream i ostream*** są deklaracje wielu funkcji składowych do obsługi strumieni.

Dość popularne funkcje to:

-int get(); deklarowana w klasie ***istream***

oraz

-ostream & put (char) deklarowana w klasie ***ostream***

Funkcja get() pobiera znak ze strumienia wejściowego, funkcja put(char) wstawia

Wejście/wyjście

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

int main()
{
   char znak;
   cout <<"podaj znaki, * konczy sekwencje, ENTER konczy "<<endl;
   while ((znak = cin.get()) != '*')
       cout.put(znak);
   cout<<endl;
   return 0;
}
//-------------------------------------------------------------------------------
```

Wynik:

```
podaj znaki, * konczy sekwencje, ENTER konczy
asdf*ghj
asdf
```

W programie znak '*' jest terminatorem, możemy z klawiatury wprowadzać kolejne znaki (ENTER kończy wprowadzanie), ale na ekranie pojawią się tylko znaki wprowadzone przed terminatorem.

Kolejny program możliwości get(), peek() i putback().

Wejście/wyjście

```
#include <iostream>

using namespace std;

int main()
{  char z,bu[5];
   cout <<"podaj znaki, ENTER konczy "<<endl;
   cin.get(bu, 5, '\n');
   cin.putback(bu[2]);
   z = cin.peek();
   for(int i = 0; i < 5; i++)  cout.put(bu[i]);
   cout <<endl;
   cout << z << endl;

   return 0;
}
```

Wynik:

```
podaj znaki, ENTER konczy
qwerty
qwer
e
```

Wejście/wyjście

W pokazanym programie wykorzystaliśmy funkcję get();

istream & get(char * buf, int num, char delim = '\n')

Ta funkcja wczytuje znaki i umieszcza je w tablicy buf, czyta n znaków chyba ,ze napotka znak delimitera. Parametr int num jest opcjonaly, gdy go nie podamy, funkcja czyta znaki aż napotka znak delimitera. Funkcja getline() ma syntaks:

istream & getline(char * buf, int num, char delim = '\n')

Widzimy , że argumenty sa takie same jak funkcji get(). Jaka jest różnica? Funkcja getline() pobiera i usuwa znak kończący wprowadzanie danych ze strumienia wejściowego.

Funkcja put back() ma prototyp:

istream & putback(char z)

Jest to trochę dziwna funkcja – zwraca ostatnio pobrany znak do to tego samego strumienia danych z których ów znak był pobrany !

Specjalistyczna funkcja peek():

int istream :: peek()

Przekazuje następny znak ze strumienia, bez usuwania go.

Do dyspozycji mamy także dwie analogiczne funkcje strumieniowe:

istream & read (char * buf, int num)

istream & write (char * buf, int num)

Wejście/wyjście

```
#include <iostream>
#include <cstring>  //strlen()

using namespace std;

int main()
{   char *kraj1 = "Polska";
    char *kraj2 = "Ukraina";
    int n1 = strlen(kraj1);
    int n2 = strlen(kraj2);
    cout.write(kraj1, n1);
    cout <<endl;
    for (int i =0; i <= n2; i++){
        cout.write(kraj2,i) ;
        cout <<endl;
    }
    cout <<"przekroczony limit:"<<endl;
     cout.write(kraj1, n1 + 4);
     cout <<endl;
    return 0;
}
```

Wynik:

```
Polska

U
Uk
Ukr
Ukra
Ukrai
Ukrain
Ukraina
przekroczony limit:
Polska Ukr
```

Uwaga!

W linii:

```
    cout.write(kraj1, n1 + 4);
```

Jest nieuprawnione zlecenie

Wejście/wyjście

```
#include <iostream>
const int n1 = 14;
const int n2 = 10;
using namespace std;
inline czytaj(){while (cin.get() != '\n') continue; }
int main()
{   char nazwisko[n1];
    char profesja[n2];
     cout <<"jakie nazwisko? \n";
        cin.get(nazwisko,n1 );
    if(cin.peek() != '\n')
        cout <<"dlugie nazwisko: "<<nazwisko<<endl;
    czytaj();
    cout <<"jaka profesj? \n";
    cin.get(profesja,n2 );
    if(cin.peek() != '\n')
        cout <<"dluga profesja: "<<profesja<<endl;
    cout <<"\nnazwisko: "<<nazwisko
         <<"\nprofesja: "<<profesja<<endl;
    cout <<endl;
    return 0;
}
```

Wynik:

```
jakie nazwisko?
Jan Kowalski - Hulajnoga
dlugie nazwisko: Jan Kowalski
jaka profesj?
kierowca aut spalinowych
dluga profesja: kierowca

nazwisko: Jan Kowalski
profesja: kierowca
```

Funkcja peek() sprawdza, czy został odczytany cały wiersz. Gdy w tablicy mieści się tylko część wiersza, program usuwa ze strumienia resztę

Wejście/wyjście - pliki

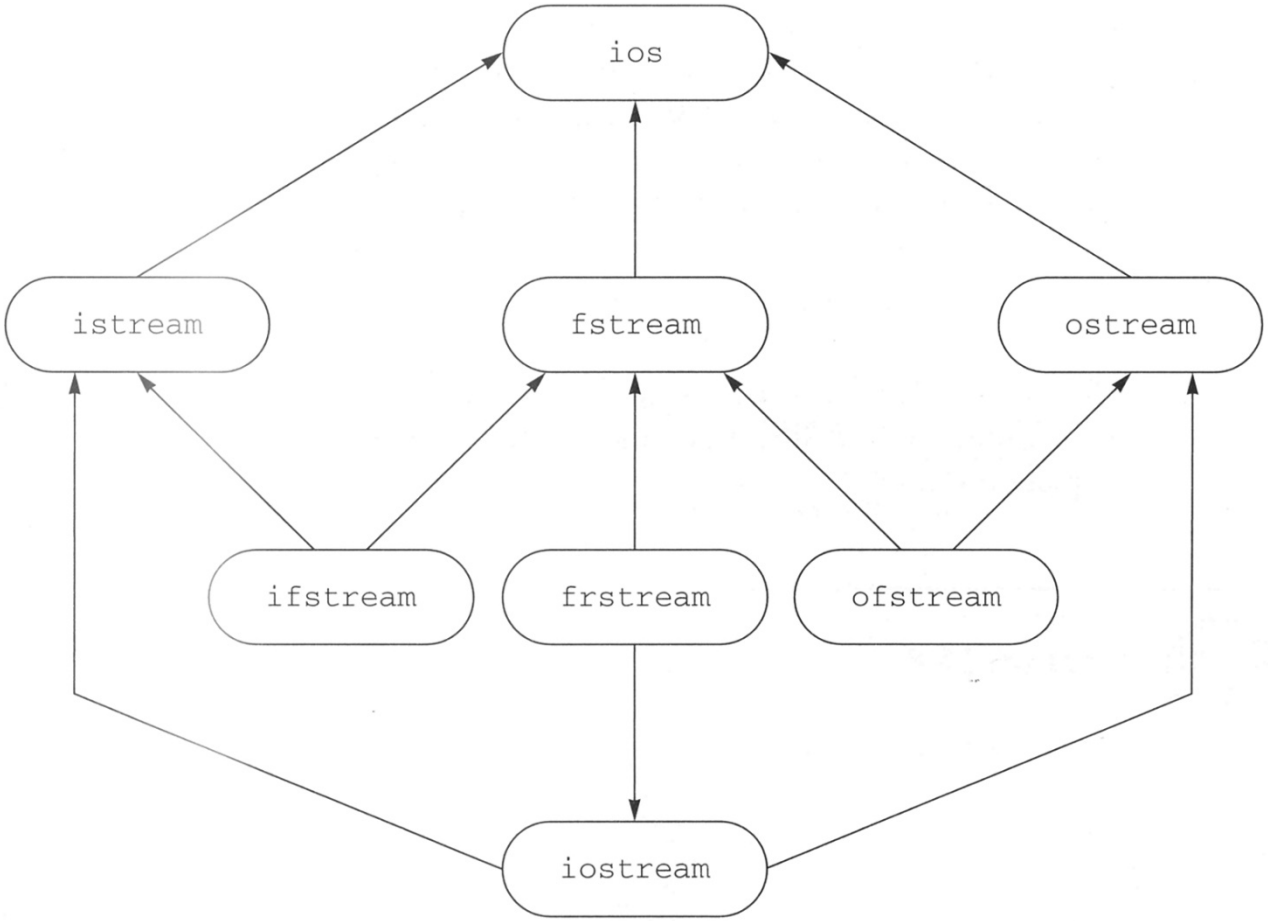
W praktyce programy komputerowe korzystają z plików. Biblioteki języka C++ realizują obsługę wejścia/wyjścia plikowego podobnie jak obsługę wejścia/wyjścia standardowego. Jeżeli chcemy zrealizować zapis do pliku, tworzymy obiekt klasy ***ofstream***.

Aby odczytać dane z pliku musimy utworzyć obiekt klasy ***ifstream***. Gdy są potrzebne operacje wykonywane wewnątrz pliku wykorzystujemy klasę ***ifstream***.

Mamy do dyspozycji także klasę ***fstream*** do jednoczesnej obsługi operacji wejścia/wyjścia.

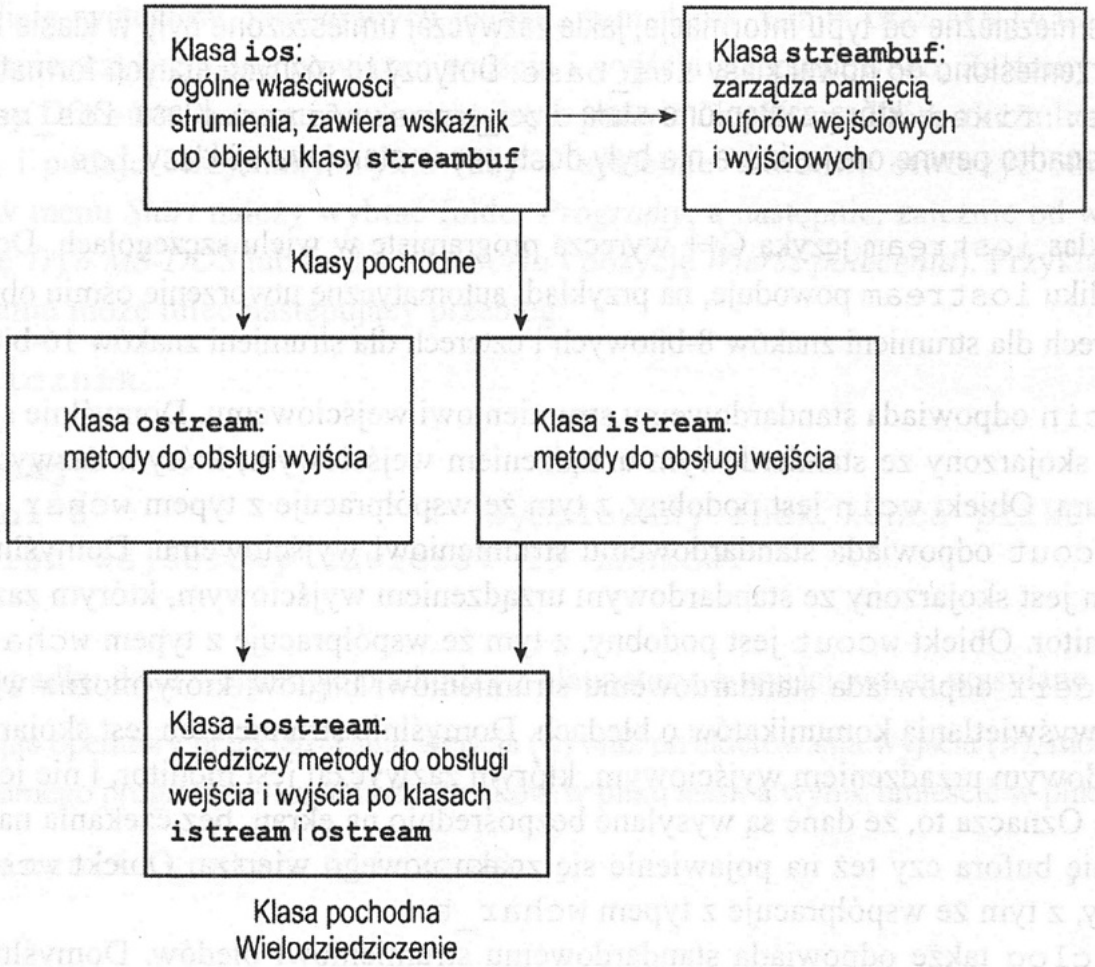
Wejście/wyjście - pliki

Hierarchia klas obsługi plików



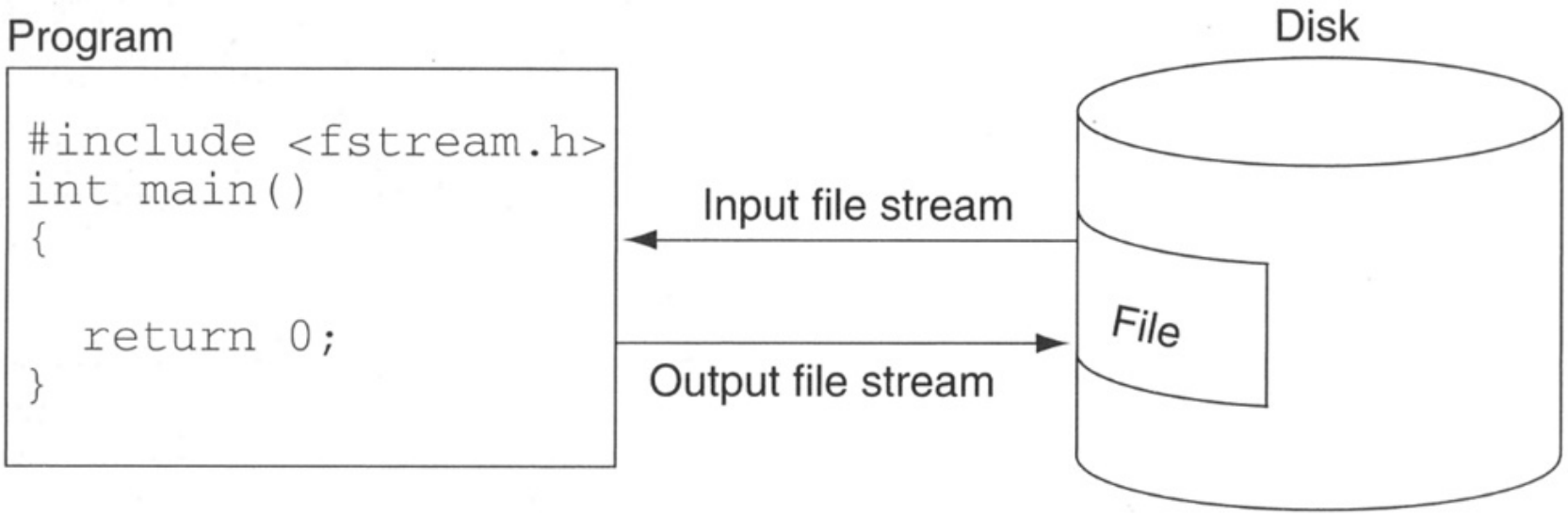
Wejście/wyjście - pliki

Niektóre klasy wejścia/wyjścia



Wejście/wyjście - pliki

Dowolna kolekcja danych, która jest zapamiętana pod wspólna nazwą na nośniku pamięci jest nazywana plikiem danych (ang. data file). Program komputerowy, zapamiętany na dysku jest przykładem pliku. Komunikacja miedzy plikami odbywa się dzięki strumieniom danych.
Na rysunku mamy ilustrację przepływu danych. Wyróżniamy strumień wejściowy (input file stream) oraz strumień wyjściowy (output file stream).



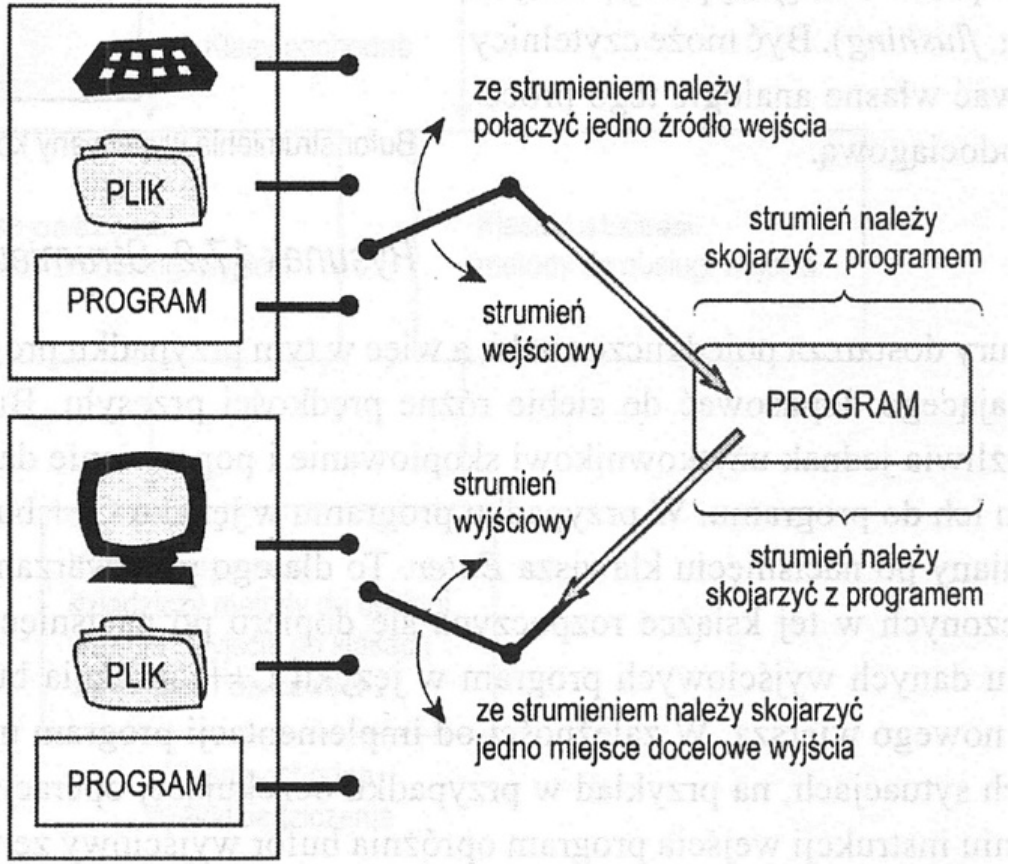
Obiekt strumienia wejściowego jest typu ***ifstream***, obiekt strumienia wyjściowego jest typu ***ofstream***

Wejście/wyjście - pliki

Obsługa wejścia składa się z dwóch procesów:

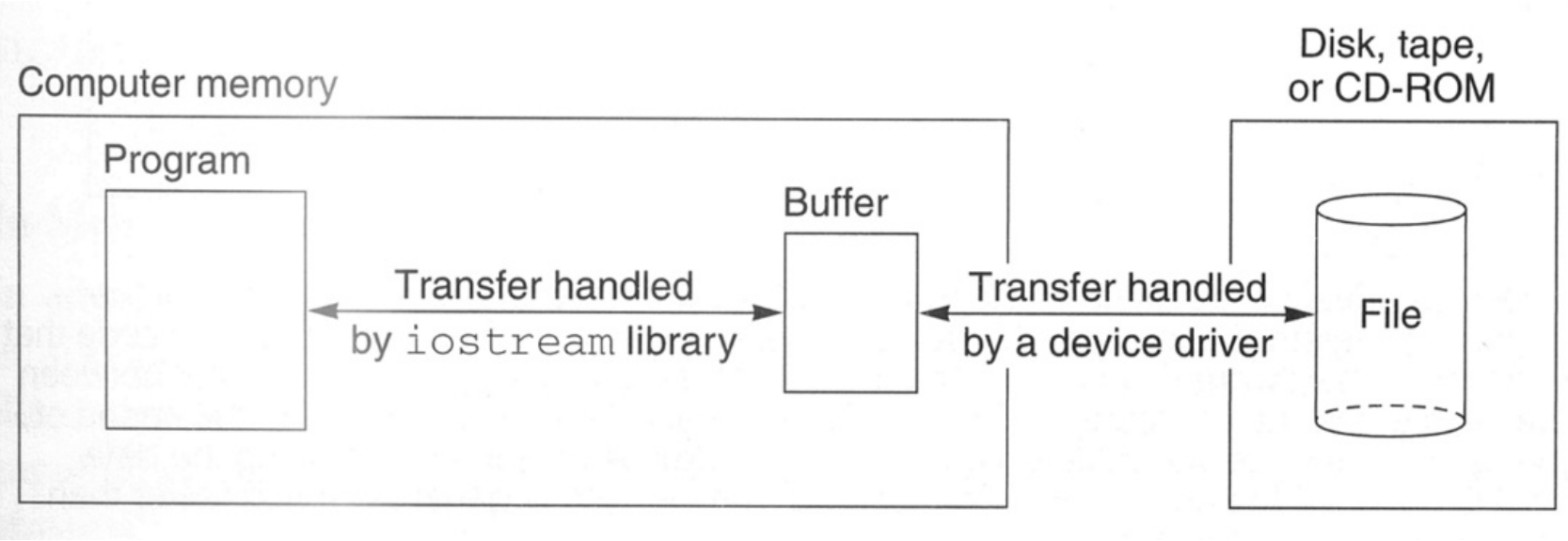
- skojarzenia strumienia z wejściem do programu
- połączenia strumienia z plikiem

Podobnie opisujemy obsługę wyjścia,



Wejście/wyjście - pliki

Obsługa wejścia wyjścia może być usprawniona dzięki wykorzystaniu bufora. Bufor jest blokiem pamięci. Urządzenia takie jak dyski często wysyłają dane w paczkach np. po 512 bajtów, podczas gdy program może operować na poszczególnych bajtach. Bufor przyjmuje określoną liczbę bajtów a na żądanie może wysłać je do miejsca docelowego. Taka operacja znacznie usprawnia sterowanie strumieniem danych (znacznie przyspiesza proces). Poniżej rysunek ilustruje mechanizm transferu danych z wykorzystaniem bufora.



Zarządzanie strumieniami i buforami jest skomplikowane. Na szczęście biblioteczny plik ***iostream*** udostępnia kilka klas dzięki którym programista ma bardzo ułatwione zadanie. W klasie ***ios*** jest zadeklarowany wskaźnik do klasy ***streambuf***, która obsługuje klasy buforów strumieni.

Wejście/wyjście - pliki

Często chcemy zapisać dane generowane przez nasz program na trwałym nośniku – wybieramy zapis na dysku twardym (mamy pewność, że nasze dane będą zachowane przez wystarczająco długi czas).

Aby osiągnąć ten cel należy wykonać następujące operacje:

- utworzyć obiekt klasy ***ofstream*** (obsługa strumienia wyjściowego)
- skojarzyć ten obiekt z konkretnym plikiem (np. dajemy nazwę ***fout***)
- obiekt (***fout***) wysyła dane do pliku, podobnie jak obiekt ***cout*** wysyła dane na ekran

Najpierw musimy włączyć odpowiednie pliki biblioteczne do naszego programu. Włączony plik ***fstream*** automatycznie dołączy plik ***iostream***. Rekomenduje się następnie utworzyć obiekt klasy ***ofstream***:

```
ofstream fout;
```

Nazwa taka jak ***fout*** może być dowolna (inne np. : fwy, outplik. flola, itp.). Kolejno należy nawiązać kontakt z plikiem. Do tego celu służy metoda ***open().*** Musimy także podać nazwę pliku docelowego (np. tekst1.txt):

```
fout.open( ”tekst1.txt” );
```

Po tych przygotowaniach polecenie typu:

```
fout <<”proste programowanie”;
```

spowoduje umieszczenia na dysku naszego komunikatu w wyspecyfikowanym pliku.

Wejście/wyjście - pliki

Często chcemy zapisać dane na dysku odczytać i wprowadzić do naszego programu.

Aby osiągnąć ten cel należy wykonać następujące operacje:

- utworzyć obiekt klasy ***ifstream*** (obsługa strumienia wejściowego)
- skojarzyć ten obiekt z konkretnym plikiem (np. dajemy nazwę ***fin***)
- obiekt (***fin***) wysyła dane do programu, podobnie jak obiekt ***cin*** pobiera dane z klawiatury

Najpierw musimy włączyć odpowiednie pliki biblioteczne do naszego programu. Włączony plik ***fstream*** automatycznie dołączy plik ***iostream***. Rekomenduje się następnie utworzyć obiekt klasy ***ifstream***:

```
ifstream fin;
```

Kolejno należy nawiązać kontakt z plikiem. Do tego celu służy metoda ***open().*** Musimy także podać nazwę pliku docelowego (np. tekst1.txt):

```
fin.open( ”tekst1.txt” );
```

Po tych przygotowaniach polecenia typu:

```
char buf[20];
fin >> buf;
fin.getline(buf,20);
```

spowodują odczytanie danych z pliku dyskowego.

Wejście/wyjście - pliki

```
#include <fstream>
//#include <iostream>
using namespace std;

int main()
{   ofstream fout;
    fout.open("C:\\Users\\mikfiz\\test1.txt");
    fout <<"program w C++";
    fout.close();
    return 0;
}
```

Metoda open() ma postać:
void open(char* nazwa, int tryb, int dostep)
nazwa to nazwa pliku
tryb – określa sposób otwarcia pliku
dostep – określa prawa dostępu do pliku

Pokazany program tworzy plik o nazwie ***test1.txt*** (podajemy także ścieżkę dostępu).

W pliku zastaje umieszczony napis „program w C++”. Zakładamy milcząco, że wszystko odbyło się prawidłowo. Przy pomocy Notatnika (Windows) możemy w wyspecyfikowanym katalogu odszukać nasz plik i wyświetlić jego zawartość.

Wejście/wyjście - pliki

```
#include <fstream>
#include <iostream>
using namespace std;
int main()
{   char z;
    ifstream fin;
    fin.open("C:\\Users\\mikfiz\\test1.txt");
    while(fin.get(z))
        cout <<z;
    cout <<endl;
    fin.close();
    return 0;
}
```

Wynik:

program w C++

W tym programie pobieramy dane z pliku test1.txt. Plik test1.txt jest umieszczony na dysku C. Zawartość pliku wyświetlana jest na ekranie monitora. Zakładamy, że wszystkie operacje są poprawne. Funkcja close() zamyka plik.

Wejście/wyjście - pliki

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

int main()
{   char z;
    char* s1 = "programowanie w C++ (s1)";
    char* s2 = "nie jest trudne (s2)\n";
    char* s3 = "zasadniczo (s3";
    ofstream os1("C:\\Users\\mikfiz\\test1.txt");   //plik test1
    ofstream os2("C:\\Users\\mikfiz\\test2.txt");   //plik test2
    ofstream os3("C:\\Users\\mikfiz\\test3.txt");   //plik test3
    os1 << s1;                                      // w test1 zapisuje s1
    os1.close();
    os2 << s2 << s3;                                //w test2 zapisuje s2 + s3
    os2.close();
    ifstream is1("C:\\Users\\mikfiz\\test2.txt");    //kontakt z test2
    while(os3 && is1.get(z)) os3.put(z);            //w test3 kopiuje test2
    is1.close();
    os3.close();
    return 0;
}
```

Program obsługuje 3 pliki

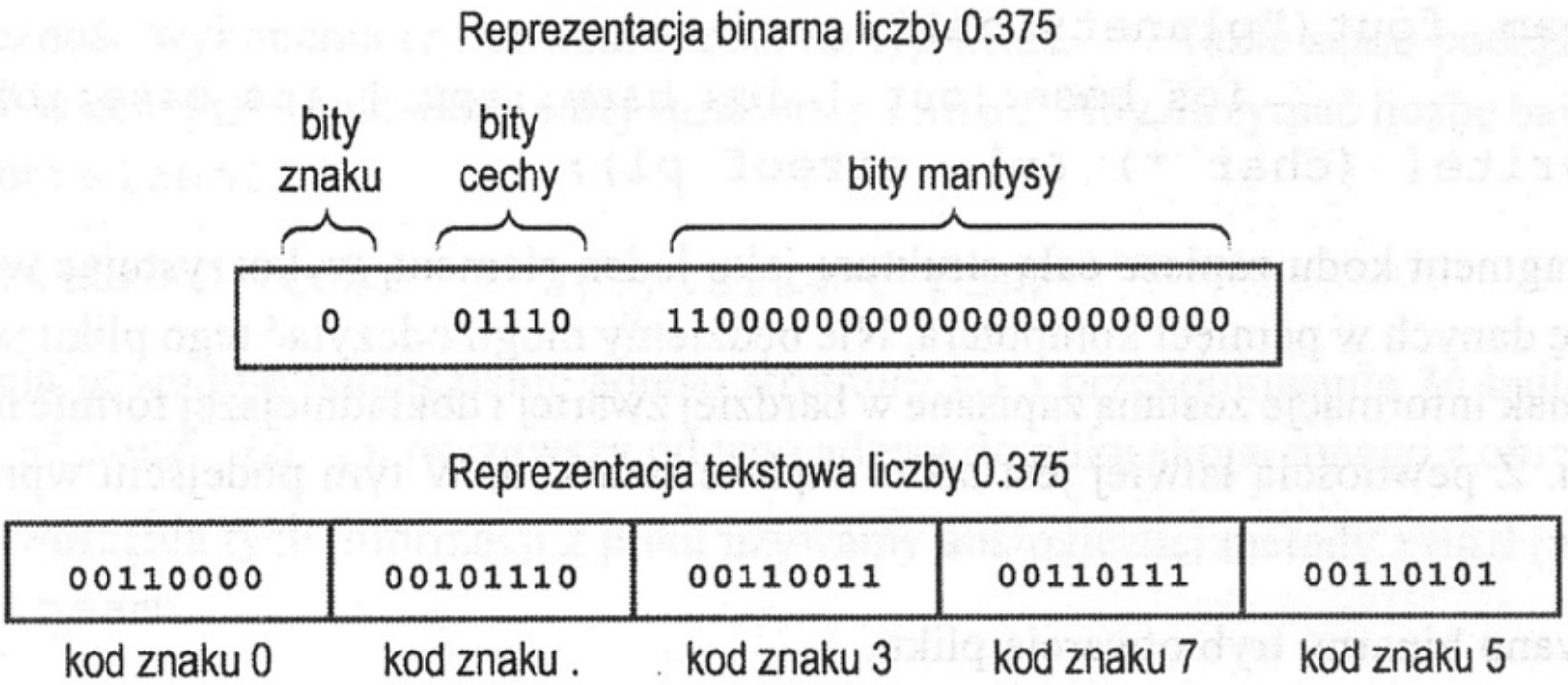
test1:
programowanie w C++ (s1)

test2:
nie jest trudne (s2)
zasadniczo (s3

test3:
nie jest trudne (s2)
zasadniczo (s3

Wejście/wyjście - pliki

Dane w systemach komputerowych możemy zapisywać albo w postaci tekstowej (obecnie popularna postać) albo w postaci binarnej. Zapis tekstowy oznacza, że wszystko zapisane jest w postaci znaków, binarnie – reprezentacja w czystym kodzie binarnym. To oznacza , że pliki zapisane w trybie binarnym są znacznie mniejsze.



Zadanie 1

Z1.Utwórz program zawierający cztery tablice. Trzy tablice mają zawierać:

- imię
- drugie imię
- nazwisko

Skopiuj wymienione trzy ciagi (z trzech tablic) do czwartej tablicy w celu otrzymania pełnego imienia i nazwiska.

Korzystać z funkcji strncpy() i strcpy()

Zadanie 2

Z2. Napisz program zliczający znaki w danych wejściowych az do napotkania znaku $, który powinien pozostać w strumieniu wejściowy.

KONIEC