Zbiór zadań z C/C++

Przemysław Olbratowski

7 lipca 2013

Ten dokument to zbiór akademickich zadań z języka C/C++ oraz bibliotek STL i Qt. Napisałem go prowadząc przez kilka lat ćwiczenia z programowania dla najlepszych studentów Wydziału Fizyki Uniwersytetu Warszawskiego. Wszystkie zadania opatrzone są przygotowanymi przeze mnie rozwiązaniami. Zbiór ten jest ciągle rozwijany i może być niekompletny.

Spis treści

1 Narzędzia programistyczne

1.1 Make

1.1.1 Make

1. Napisz program `kwadrat` wczytujący ze standardowego wejścia liczbę rzeczywistą i wypisujący na standardowe wyjście jej kwadrat. Niech program składa się z dwóch modułów - `kwadrat.cc` zawierającego funkcję `main` oraz `modul.cc` zawierającego funkcję obliczającą kwadrat.

2. Przygotuj plik `makefile` do kompilacji i łączenia tego programu:

   - przy pomocy reguł jawnych;
   - definiując i wykorzystując w pliku odpowiednie domyślne reguły kompilacji i łączenia;
   - wykorzystując predefiniowane domyślne reguły kompilacji, a łączenie przeprowadzając przy pomocy reguł jawnych.

   Plik `makefile` powinien być sporządzony tak, aby zmiana któregokolwiek z modułów, a także pliku nagłównkowego, powodowała przekompilowanie tylko odpowiedniego modułu oraz połączenie modułów w plik wykonywalny.

3. Przygotuj plik `makefile` w taki sposób, aby kompilację umożliwiającą późniejsze debuggowanie można było osiągnąć przez odpowiednią zmianę wartości zmiennej `CXXFLAGS`.

4. Dodaj regułę jawną do kompilacji pliku `hymn.rex` przy pomocy polecenia `pdflatex`.

5. Dodaj regułę jawną usuwającą z katalogu wszystkie pliki tworzone przez program `make` przy przetwarzaniu obecnego pliku `makefile`.

6. Przygotuj końcowy plik `makefile` tak, aby:

   - polecenie `make` tworzyło plik wykonywalny `kwadrat` i plik `hymn.pdf`;
   - polecenie `make clean` usuwało wszystkie pliki tworzone komendą `make`.

   Oba te polecenia powinny działać niezależnie od obecności w katalogu innych plików o dowolnych nazwach.

1.2 GDB

1.2.1 Bug

Dany jest następujący program `bug.cc`:

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 int i=10;
 6
 7 int main ()
 8 {
 9   for (int i=3;i<7;i++)
10     cout<<i<<endl;
```

```
11   cout<<i<<endl;
12   return i;
13 }
```

1. Co zostanie wypisane na ekran w wyniku działania tego programu?
2. Jaki wynik da polecenie `echo $?` wydane zaraz po wykonaniu programu?
3. Program ten wczytano do debuggera `gdb`. Jaki będzie rezultat następujących sekwencji poleceń? Każda sekwencja podawana jest bezpośrednio po uruchomieniu debuggera.

   - ```
     break 12
     run
     print i
     print ::i
     continue
     ```
   - ```
     break 9
     run
     step
     step
     print i
     ```
   - ```
     break 10 if i==5
     run
     set var i=10
     continue
     ```

   Odpowiedź podaj bez uciekania się do metod empirycznych.

1.2.2 Factorial

Początkujący programista chce obliczyć sumę odwrotności silni liczb od 0 do 10. Napisał w tym celu taki program `factorial.cc`:

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 int main ()
 6 {
 7   int n,s;
 8   for (n=0;n<10;n++)
 9     s+=1/factorial (n);
10   cout<<s<<endl;
11   return 0;
12 }
```

Funkcja `factorial` znajduje się w osobnym pliku. Niestety w czasie wykonania programu wyświetlany jest komunikat o błędzie. Bardziej doświadczony kolega wczytuje program do debuggera `gdb` i wydaje następujące polecenia:

```
break 9 if !n
run
```
```

W odpowiedzi dostaje komunikat:

```
Breakpoint 1, main () at factorial.cc:9
9           s+=1/factorial (n);
```

Na czym polega błąd w programie? W programie tym są jeszcze cztery inne błędy. Znajdź je.

1.3 Gnuplot

1.3.1 Data

Niniejszy przykład pokazuje jak sporządzać wykresy wielkości obliczonych przez programy w `C++` przy pomocy programu `gnuplot`.

Przypuśćmy, że chcemy sporządzić wykres funkcji $\sin x + 2\cos 2x$ w przedziale od -5 do 5 próbkując funkcję z krokiem 0.1. Piszemy w `C++` następujący program `data.cpp`:

```
#include <iostream>
#include <cmath>

using namespace std;

int main () {
  for (double x=-5;x<5;x+=0.1)
    cout<<x<<" "<<sin (x)+2*cos (2*x)<<endl;
  return 0; }
```

Kompilujemy go komendą:

```
g++ -o data data.cpp
```

Program ten wypisuje na standardowe wyjście dwie kolumny - w pierwszej kolejne odcięte, a w drugiej odpowiadające im wartości funkcji. Oto kilka pierwszych linii tego wydruku:

```
-5 -0.719219
-4.9 -0.8784
-4.8 -0.973211
-4.7 -0.999463
-4.6 -0.955996
-4.5 -0.84473
```

Przekierowujemy teraz wydruk do pliku `function.dat` pisząc:

```
./data > function.dat
```

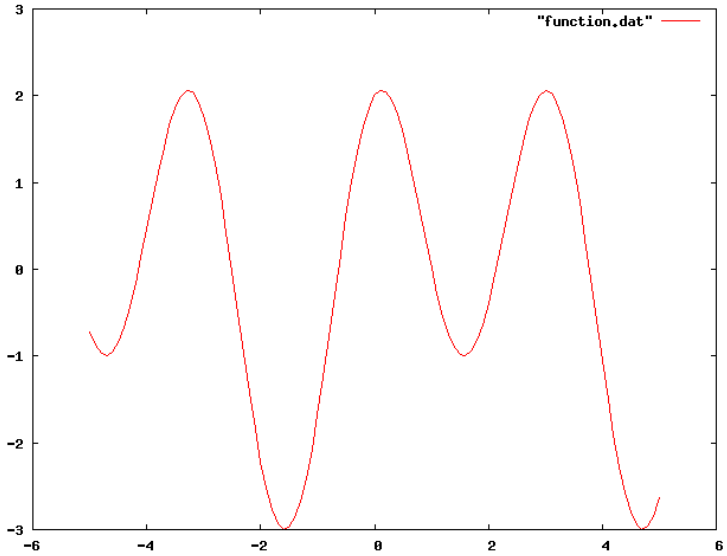
W katalogu, w którym znajduje się ten plik, uruchamiamy program `gnuplot` poleceniem

```
gnuplot
```

Aby sporządzić wykres, wydajemy w programie `gnuplot` polecenie

```
plot "function.dat" with lines
```

Uzyskujemy w ten sposób następujący wykres:



2 Podstawowe elementy języka

2.1 Operacje wejścia-wyjścia. Zmienne.

2.1.1 Hello

Napisz program `hello` wypisujący napis `Hello world!`.

```
#include <iostream>

using namespace std;

int main () {
  cout<<"Hello world!"<<endl;
  return 0; }
```

2.1.2 Variables

Napisz interaktywny program `variables`, który pyta użytkownika o imię i wiek, a następnie wypisuje komunikat informujący, w którym roku urodził się użytkownik.

```
#include <iostream>
#include <string>

using namespace std;

int main () {
  string imie;
  int wiek;
  cout<<"Jak masz na imie? ";
  cin>>imie;
  cout<<Ile masz lat? ";
  cin>>wiek;
```

```
  cout<<imie<<" urodzil sie w roku "<<2012-wiek<<'.'<<endl;
  return 0; }
```

2.2 Instrukcje iteracyjne. Formatowanie wydruku.

2.2.1 Loops

Niniejszy przykład ilustruje działanie trzech istniejących w języku `C++` instrukcji iteracyjnych: `for`, `while` i `do while`. Każdy z poniższych wycinków programu wypisuje na standardowe wyjście cyfry od `0` do `9`.

- Przy pomocy pętli `for`

```
for (int i=0;i<10;++i)
  cout<<i<<endl;
```

- Przy pomocy pętli `while`

```
int i=0;
while (i<10) {
  cout<<i<<endl;
  ++i; }
```

- Przy pomocy pętli `do while`

```
int i=0;
do {
  cout<<i<<endl;
  ++i; }
while (i<10);
```

2.2.2 Factorial

Napisz program `factorial`, który wczytuje ze standardowego wejścia liczbę naturalną i wypisuje na standardowe wyjście jej silnię. Pamiętaj, że $0! = 1$.

```
#include <iostream>

using namespace std;

int main () {
  int n,k=1;
  cin>>n;
  for (int i=2;i<=n;i++)
    k*=i;
  cout<<k<<endl;
  return 0; }
```

2.2.3 Factor

Daną liczbę naturalną można rozłożyć na czynniki pierwsze w następujący sposób. Sprawdzamy, czy liczba dzieli się przez 2. Jeżeli tak, to stwierdzamy, że dwójka występuje w jej rozkładzie na czynniki pierwsze, a samą liczbę dzielimy przez dwa. Czynność tę powtarzamy aż liczba przestanie być podzielna przez dwa. Następnie powtarzamy tę samą procedurę badając podzielność przez 3, 4 i tak dalej, aż rozważana liczba stanie się równa 1.

Napisz program `factor`, który wczytuje ze standardowego wejścia liczbę naturalną, a następnie wypisuje na standardowe wyjście jej rozkład na czynniki pierwsze.

Rozwiązanie 1

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

int main () {
  int n;
  cin>>n;
  int i=2;
  while (n!=1)
    if (n%i)
      i++;
    else {
      cout<<i<<endl;
      n/=i; }
  return 0; }
```

Rozwiązanie 2

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

int main () {
  int n;
  cin>>n;
  for (int i=2;n!=1;i++)
    while (!(n%i))
      cout<<i<<endl;
      n/=i; }
  return 0; }
```

2.2.4 Format

Napisz program `format` wypisujący w kolejnych wierszach pierwszych 15 potęg liczby π poprzedzonych numerem potęgi

1. bez formatowania wydruku.
2. z dokładnością do trzech cyfr znaczących.
3. z dokładnością do trzech cyfr po przecinku w taki sposób, aby wszystkie kropki dziesiętne znalazły się w jednej kolumnie. Numer potęgi powinien być sformatowany tak, aby wszystkie cyfry jednostek wypadały w jednej kolumnie.

4. w formacie naukowym z dokładnością do trzech cyfr znaczących w taki sposób, aby wszystkie kropki dziesiętne znalazły się w jednej kolumnie. Numer potęgi jak w poprzednim punkcie.

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

int main () {
  double x;
  x=1;
  for (int i=0;i<15;i++) {
    cout<<i<<" "<<x<<endl;
    x*=M_PI; }
  cout<<endl;
  x=1;
  cout<<setprecision (3);
  for (int i=0;i<15;i++) {
    cout<<i<<" "<<x<<endl;
    x*=M_PI; }
  cout<<endl;
  x=1;
  cout<<fixed<<setprecision (3);
  for (int i=0;i<15;i++) {
    cout<<setw (2)<<i<<" "<<setw (11)<<x<<endl;
    x*=M_PI; }
  cout<<endl;
  x=1;
  cout<<scientific<<setprecision(2);
  for (int i=0;i<15;i++) {
    cout<<setw(2)<<i<<" "<<setw(9)<<x<<endl;
    x*=M_PI; }
  return 0; }
```

Wykonanie tego programu daje następujący wynik (dla wygody poszczególne wydruki umieszczono obok siebie):

```
0 1                0 1                  0       1.000       0  1.00e+00
1 3.14159          1 3.14               1       3.142       1  3.14e+00
2 9.8696           2 9.87               2       9.870       2  9.87e+00
3 31.0063          3 31                 3      31.006       3  3.10e+01
4 97.4091          4 97.4               4      97.409       4  9.74e+01
5 306.02           5 306                5     306.020       5  3.06e+02
6 961.389          6 961                6     961.389       6  9.61e+02
7 3020.29          7 3.02e+03           7    3020.293       7  3.02e+03
8 9488.53          8 9.49e+03           8    9488.531       8  9.49e+03
9 29809.1          9 2.98e+04           9   29809.099       9  2.98e+04
10 93648           10 9.36e+04         10   93648.047      10  9.36e+04
```

```
11 294204           11 2.94e+05         11  294204.018      11  2.94e+05
12 924269           12 9.24e+05         12  924269.182      12  9.24e+05
13 2.90368e+06      13 2.9e+06          13 2903677.271      13  2.90e+06
14 9.12217e+06      14 9.12e+06         14 9122171.182      14  9.12e+06
```

2.2.5 Multiplication

Napisz program `multiplication.cc`, który wczytuje ze standardowego wejścia liczbę naturalną `n` i wypisuje na standardowe wyjście tabliczkę mnożenia liczb od 1 do `n`. Tabliczka powinna być sformatowana tak, jak w poniższym przykładzie dla `n=10`:

```
      1   2   3   4   5   6   7   8   9  10
    ---------------------------------------
  1|   1   2   3   4   5   6   7   8   9  10
  2|   2   4   6   8  10  12  14  16  18  20
  3|   3   6   9  12  15  18  21  24  27  30
  4|   4   8  12  16  20  24  28  32  36  40
  5|   5  10  15  20  25  30  35  40  45  50
  6|   6  12  18  24  30  36  42  48  54  60
  7|   7  14  21  28  35  42  49  56  63  70
  8|   8  16  24  32  40  48  56  64  72  80
  9|   9  18  27  36  45  54  63  72  81  90
 10|  10  20  30  40  50  60  70  80  90 100
```

Szerokości kolumn powinny być dobierane automatycznie na podstawie wartości `n`.

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

int main () {
  int n,w;
  cin>>n;
  w=log10 (n*n)+1;
  cout<<setw (w)<<""<<" ";
  for (int j=1;j<=n;j++)
    cout<<" "<<setw (w)<<j;
  cout<<endl;
  cout<<setw (w)<<""<<" ";
  cout<<setfill ('-');
  for (int j=1;j<=n;j++)
    cout<<"-"<<setw (w)<<"";
  cout<<setfill (' ');
  cout<<endl;
  for (int i=1;i<=n;i++) {
  cout<<setw (w)<<i<<"|";
    for (int j=1;j<=n;j++)
      cout<<" "<<setw (w)<<i*j;
    cout<<endl; }
```

```
  return 0; }
```

2.3 Instrukcja warunkowa. Tablice

2.3.1 Prime

Napisz program `prime` sprawdzający, czy dana liczba naturalna jest pierwsza. Program powinien wczytywać liczbę ze standardowego wejścia i drukować na standardowe wyjście odpowiedni komunikat.

```
#include <iostream>
#include <cmath>

using namespace std;

int main () {
  int n,i;
  cin>>n;
  for (i=2;i<sqrt (n)+0.5;i++)
    if (!(n%i)) {
      cout<<"No"<<endl;
      return 0; }
  cout<<"Yes"<<endl;
  return 0; }
```

2.3.2 Mini

Napisz program `mini` znajdujący położenie najmniejszego elementu tablicy. Program powinien wczytywać długość tablicy, tworzyć tę tablicę w pamięci, wypełniać losowymi liczbami rzeczywistymi z przedziału od 0 do 1, a następnie wypisywć elementy tablicy wraz z indeksami oraz znaleziony indeks i wartość elementu najmniejszego.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main () {
  srand (time (0));
  int num;
  cin>>num;
  double vec [num];
  for (int cnt=0;cnt<num;cnt++)
    cout<<(vec [cnt]=(double) rand ()/RAND_MAX)<<endl;
  cout<<endl;
  int min=0;
  for (int cnt=1;cnt<num;cnt++)
    if (vec [cnt]<vec [min])
      min=cnt;
  cout<<min<<" "<<vec [min]<<endl;
```

```
  return 0; }
```

2.3.3 Next

Napisz program `next` znajdujący położenie drugiego co do wielkości elementu tablicy. Program powinien wczytywać długość tablicy, tworzyć tę tablicę w pamięci, wypełniać losowymi liczbami rzeczywistymi z przedziału od 0 do 1, a następnie wypisywć elementy tablicy wraz z indeksami oraz znaleziony indeks i wartość elementu drugiego co do wartości.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main () {
  srand (time (0));
  int num;
  cin>>num;
  double vec [num];
  for (int cnt=0;cnt<num;cnt++)
    cout<<(vec [cnt]=(double) rand ()/RAND_MAX)<<endl;
  cout<<endl;
  int min,nex;
  if (vec [0]<vec [1]) {
    min=0; nex=1; }
  else {
    min=1; nex=0; }
  for (int ind=2;ind<num;ind++)
    if (vec [ind]<vec [min]) {
      nex=min; min=ind; }
    else if (vec [ind]<vec [nex])
      nex=ind;
  cout<<nex<<" "<<vec [nex]<<endl;
  return 0; }
```

2.3.4 Sequence - zadanie z gwiazdką

Napisz program `sequence` znajdujący w zadanej tablicy całkowitej takie dwa elementy, dla których suma elementów od pierwszego do drugiego włącznie jest największa. Program powinien wczytywać ze standardowego wejścia długość tablicy, tworzyć ją w pamięci, wypełniać losowymi liczbami całkowitymi z przedziału od -5 do 5 włącznie i wypisywać tę tablicę na ekran. Następnie program powinien znajdować i wypisywać indeksy szukanych dwóch elementów oraz odpowiadającą im sumę.

```
#include <climits>
#include <cstdlib>
#include <ctime>
#include <iostream>
```

```
using namespace std;

int main () {
  srandom (time (0));
  int num;
  cin>>num;
  double vec [num];
  for (int i=0;i<num;i++)
    vec [i]=random ()%11-5;
  for (int i=0;i<num;i++)
    cout<<vec [i]<<" ";
  cout<<endl;
  int ini=0,fin=0,max=INT_MIN;
  int i=0,s=0;
  for (int j=0;j<num;j++) {
    s+=vec [j];
    if (s>max) {
      ini=i;
      fin=j;
      max=s; }
    if (s<=0) {
      i=j+1;
      s=0; }}
  cout<<ini<<" "<<fin<<" "<<max<<endl;
  return 0; }
```

2.3.5 Eratostenes

Napisz program `eratostenes` znajdujący metodą sita Eratostenesa wszystkie liczby pierwsze mniejsze od danej liczby naturalnej. Program powinien czytać tę liczbę ze standardowego wejścia, a wynik wypisywać na standardowe wyjście.

Eratostenes z Cyreny, ur. ok. 276 p.n.e., zm. ok. 194 p.n.e., gr. filozof, astronom, matematyk i geograf; kierował słynną Biblioteką Aleksandryjską; podał sposób podwojenia sześcianu i metodę wyznacznia liczb pierwszych zw. sitem Eratostenesa; zaproponował wprowadzenie lat przestępnych; zmierzył długość południka.

```
#include <iostream>

using namespace std;

int main () {
  int n;
  cin>>n;
  bool s [n];
  for (int i=2;i<n;i++)
    s [i]=true;
  for (int i=2;i<n;i++)
    if (s [i]) {
      cout<<i<<endl;
      for (int j=2*i;j<n;j+=i)
```

```
      s [j]=false; }
  return 0; }
```

2.3.6 Goldbach

Napisz program `goldbach`, który wczytuje ze standardowego wejścia liczbę naturalną n, a następnie wypisuje na standardowe wyjście wszystkie liczby naturalne mniejsze od n, których nie da się przedstawić jako sumy dwóch liczb pierwszych. Sprawdź, czy wśród wypisanych liczb są liczby parzyste większe od dwóch.

Christian Goldbach, ur. 1690, zm. 1764, matematyk działający w Petersburgu i w Moskwie; autor prac z teorii szeregów i równań różniczkowych. W liście do matematyka szwajcarskiego Leonarda Eulera sformułował hipotezę, iż każda liczba parzysta większa od dwóch jest sumą dwóch liczb pierwszych. Przypuszczenie to, znane jako hipoteza Goldbacha, nie zostało do tej pory udowodnione ani obalone. Przy użyciu komputerów sprawdzono jego prawdziwość dla liczb sięgających około 10^{18}.

```
#include <iostream>

using namespace std;

int main () {
  int n;
  cin>>n;
  bool s [n],t [n];
  for (int i=0;i<n;i++)
    s [i]=t [i]=true;
  s [0]=s [1]=false;
  for (int i=2;i<n;i++)
    if (s [i])
      for (int j=2*i;j<n;j+=i)
        s [j]=false;
  for (int i=0;i<n;i++)
    for (int j=i;i+j<n;j++)
      if (s [i] && s [j])
        t [i+j]=false;
  for (int i=1;i<n;i++)
    if (t [i])
      cout<<i<<endl;
  return 0; }
```

2.3.7 Pascal

Napisz program `pascal` wypisujący na standardowe wyjście pierwszych 13 wierszy trójkąta Pascala sformatowanych jak poniżej:

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5  10  10   5   1
```

```
                    1      6     15     20     15      6      1
                 1      7     21     35     35     21      7      1
              1      8     28     56     70     56     28      8      1
           1      9     36     84    126    126     84     36      9      1
        1     10     45    120    210    252    210    120     45     10      1
     1     11     55    165    330    462    462    330    165     55     11      1
  1     12     66    220    495    792    924    792    495    220     66     12      1
```

Za ostatnią liczbą w każdym wierszu nie powinny być drukowane zbędne spacje.

Blaise Pascal, ur. 1623, zm. 1662, franc. matematyk, fizyk, pisarz i filozof; w wieku 16 lat udowodnił słynne twierdzenie o sześciokącie wpisanym w krzywą stożkową; pierwszy zajmował się prawdopodobieństwem z matematycznego punktu widzenia; skonstruował maszynę liczącą. Uważa się, że trójkąt Pascala został odkryty na przełomie XI i XII w. w Chinach. Pascal połączył z nim studia nad prawdopodobieństwem osiągając tak znakomite wyniki, że obecnie trójkąt ten nosi jego imię.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main () {
  const int N=13;
  int PP [N][N];
  for (int n=0;n<N;n++) {
    PP [n][0]=PP [n][n]=1;
    for (int k=1;k<n;k++)
      PP [n][k]=PP [n-1][k-1]+PP [n-1][k]; }
  for (int n=0;n<N;n++) {
    for (int k=0;k<N-n-1;k++)
      cout<<"   ";
    for (int k=0;k<n;k++)
      cout<<setw (3)<<PP [n][k]<<"   ";
    cout<<setw (3)<<PP [n][n]<<endl; }
  return 0; }
```

2.3.8 Permutation

Napisz program `permutation`, który wczytuje ze standardowego wejścia liczbę n, a następnie wypisuje na standardowe wyjście wszystkie permutacje liczb od 1 do n, jak w poniższym przykładzie dla $n = 3$:

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

```
#include <cmath>
#include <iomanip>
```

```
#include <iostream>

using namespace std;

int main () {
  int n;
  cin>>n;
  bool e [n];
  int p [n];
  for (int i=0;i<n;i++) {
    e [i]=false;
    p [i]=i; }
  int i=n-1;
  while (i>=0) {
    if (i==n-1) {
      for (int k=0;k<n;k++)
        cout<<setw (log10 (n)+2)<<1+p [k];
      cout<<endl; }
    int j;
    for (j=p [i]+1;j<n && !e [j];j++);
    if (j==n) {
      e [p [i]]=true;
      i--;
      continue; }
    e [p [i]]=true;
    e [j]=false;
    p [i]=j;
    for (j=0,i++;i<n;i++) {
      for (;!e [j];j++);
      e [j]=false;
      p [i]=j; }
    i=n-1; }}
```

2.4 Wskaźniki

2.4.1 Pointer - przykład

Poniższy przykład ilustruje wykorzystanie wskaźników do pojedynczych zmiennych.

```
#include <iostream>

using namespace std;

int main () {
  int a=2,b=7;
  cout<<a<<" "<<b<<endl;
  int *p=&a;
  cout<<p<<" "<<*p<<endl;
  *p+=b;
  cout<<p<<" "<<*p<<endl;
```

```
  p=&b;
  cout<<p<<" "<<*p<<endl;
  cout<<a<<" "<<b<<endl;
  return 0; }
```

Wykonanie tego programu daje następujący wydruk:

```
2 7
0x7fffdcee6c44 2 2
0x7fffdcee6c44 9 9
0x7fffdcee6c40 7 7
9 7
```

Oczywiście adresy zmiennych będą się różnić od wykonania do wykonania, w zależności od tego, jaki obszar pamięci zostanie programowi przydzielony przez system operacyjny.

2.4.2 Table - przykład

Poniższy przykład ilustruje wykorzystanie wskaźników w odniesieniu do tablic.

```
#include <iostream>

using namespace std;

int main () {
  int t [5]={8,3,6,5,7};
  cout<<&t [0]<<' '<<t<<endl;
  for (int i=0;i<5;i++)
    cout<<i<<' '<<&t [i]<<' '<<t+i<<' '<<t [i]<<' '<<*(t+i)<<endl;
  for (int *p=t;p<t+5;p++)
    cout<<p-t<<' '<<p<<' '<<*p<<endl;
  return 0; }
```

Wykonanie tego programu daje następujący wydruk:

```
0x7fff5fbff9f0 0x7fff5fbff9f0
0 0x7fff5fbff9f0 0x7fff5fbff9f0 8 8
1 0x7fff5fbff9f4 0x7fff5fbff9f4 3 3
2 0x7fff5fbff9f8 0x7fff5fbff9f8 6 6
3 0x7fff5fbff9fc 0x7fff5fbff9fc 5 5
4 0x7fff5fbffa00 0x7fff5fbffa00 7 7
0 0x7fff5fbff9f0 8
1 0x7fff5fbff9f4 3
2 0x7fff5fbff9f8 6
3 0x7fff5fbff9fc 5
4 0x7fff5fbffa00 7
```

Z wydruku tego można wywnioskować, że zmienne typu `int` zajmują 4 bajty.

2.4.3 Address - pytanie

Co zostanie wypisane na ekran w wyniku wykonania poniższego programu? Zadanie rozwiąż bez użycia komputera. Dokładny opis oparatora ++ można znaleźć na stronie

http://www.cplusplus.com/doc/tutorial/operators/.

```
#include <iostream>

using namespace std;

int main () {
  int v [2]={3,7};
  int *p,*q=&v [0],*r;
  p=q++;
  r=p;
  p=q;
  q=r;
  *p+=++*q;
  cout<<v [0]<<" "<<v [1]<<endl;
  return 0; }
```

2.4.4 Mini

Napisz program `mini` znajdujący najmniejszy element tablicy. Zadanie rozwiąż posługując się wyłącznie wskaźnikami, a nie indeksami elementów. Program powinien czytać ze standardowego wejścia długość tablicy, tworzyć tę tablicę w pamięci, wypełniać losowymi liczbami rzeczywistymi z przedziału od 0 do 1, a następnie wypisywć na standardowe wyjście elementy tablicy wraz z adresami oraz znaleziony adres i wartość elementu najmniejszego. Wydruk powinien być sformatowany jak w poniższym przykładzie dla trzech elementów:

```
0x7fff5fbff960 0.840
0x7fff5fbff968 0.394
0x7fff5fbff970 0.783

0x7fff5fbff968 0.394
```

```
#include <cstdlib>
#include <ctime>
#include <iostream>

using namespace std;

int main () {
  srand (time (0));
  int num;
  cin>>num;
  double vec [num];
  for (double *pnt=vec;pnt<vec+num;pnt++)
    *pnt=(double) rand ()/RAND_MAX;
  for (double *pnt=vec;pnt<vec+num;pnt++)
```

```
    cout<<pnt<<" "<<*pnt<<endl;
  cout<<endl;
  double *min=vec;
  for (double *pnt=vec+1;pnt<vec+num;pnt++)
    if (*pnt<*min)
      min=pnt;
  cout<<min<<" "<<*min<<endl;
  return 0; }
```

2.4.5 Versus - przykład

Niniejszy przykład ilustruje różnicę między tablicą a wskaźnikiem oraz konwersję tablicy do wskaźnika przy przekazywaniu argumentów do funkcji.

```
#include <iostream>

using namespace std;

void f (int t [],int *p) {
  cout<<sizeof (t)<<" "<<sizeof (p)<<endl; }

void g (int t [10],int *p) {
  cout<<sizeof (t)<<" "<<sizeof (p)<<endl; }

void h (int (*t) [10],int **p) {
  cout<<sizeof (*t)<<" "<<sizeof (*p)<<endl; }

int main () {
  int t [10];
  int *p=t;
  cout<<sizeof (t)<<" "<<sizeof (p)<<endl;
  f (t,p);
  g (t,p);
  h (&t,&p);
  return 0; }
```

Jego wykonanie daje taki wynik:

```
40 8
8 8
8 8
40 8
```

2.5 Funkcje

2.5.1 Euklides

Największy wspólny dzielnik dwóch liczb naturalnych można wyznaczyć posługując się następującym algorytmem przypisywanym Euklidesowi. Mamy dwie liczby. Większą z nich zastępujemy resztą z dzielenia przez mniejszą. Procedurę tę powtarzamy aż jedna z liczb stanie się równa zero. Wtedy druga jest poszukiwanym największym wspólnym dzielnikiem wyjściowych liczb.

Napisz funkcję `euklides` znajdującą największy wspólny dzielnik dwóch liczb naturalnych metodą Euklidesa. Deklaracja:

```
int euklides (int m,int n);
```

Argument	**Wejście**	**Wyjście**
`m`, `n`	Liczby naturalne	
`euklides`		Największy wspólny dzielnik argumentów

Napisz program `euklides`, który wczytuje ze standardowego wejścia dwie liczby naturalne i wypisuje na standardowe wyjście ich największy wspólny dzielnik.

Euklides, ur. ok. 365 p.n.e., zm. ok. 300 p.n.e., matematyk gr. pochodzący z Aten, działający gł. w Aleksandrii; autor pierwszych prac teoretycznych z matematyki; największe jego dzieło, Elementy, jest pierwszą próbą aksjomatycznego ujęcia geometrii; Elementy były podstawowym podręcznikiem geometrii do XIX w.

```
#include <iostream>

using namespace std;

int euklides (int m,int n) {
  int k;
  while (m) {
    k=n%m;
    n=m;
    m=k; }
  return n; }

int main () {
  int m,n;
  cin>>m>>n;
  cout<<euklides (m,n)<<endl;
  return 0; }
```

2.5.2 Pitagoras

Trójką pitagorejską nazywamy każde trzy liczby naturalne a, b, c, takie że $a^2 + b^2 = c^2$. Trójkę nazywamy pierwotną jeżeli liczby a, b, c są względnie pierwsze. Można pokazać, że każda trójka pierwotna jest postaci

$$a = m^2 - n^2 \qquad b = 2mn \qquad c = m^2 + n^2$$

gdzie m i n są względnie pierwszymi liczbami naturalnymi, z których jedna jest parzysta.

Wykorzystując funkcję `euklides` napisz program `pitagoras`, który wczytuje ze standardowego wejścia liczbę naturalną C, a następnie wypisuje na standardowe wyjście wszystkie pierwotne trójki pitagorejskie, dla których $c < C$.

Pitagoras z Samos, ur. ok. 580 p.n.e., zm. ok. 496 p.n.e., grecki matematyk, astronom i filozof. Twierdzenie o długościach boków w trójkącie prostokątnym przejął od matematyków babilońskich, ale prawdopodobnie jako pierwszy je udowodnił. Założył religijno-polityczny związek, który zyskał sobie później miano szkoły pitagorejskiej. Pitagorejczycy stworzyli podwaliny teorii liczb. Wiedzieli o istnieniu liczb niewymiernych, ale zobowiązani byli do zachowania tego w tajemnicy, ponieważ wedle filozofii pitagorejskiej istnienie tych liczb kłóciło się z harmonią świata.

```
#include <iostream>

using namespace std;

int main () {
  int c;
  cin>>c;
  for (int n=1;n<c;n++)
    for (int m=n+1;m*m+n*n<c;m++)
      if ((m+n)%2 && euklides (m,n)==1)
        cout<<m*m-n*n<<" "<<2*m*n<<" "<<m*m+n*n<<endl;
  return 0; }
```

2.5.3 Function - przykład

Poniższy przykład ilustruje przekazywanie argumentów do funkcji przez wartość i przez wskaźnik.

```
#include <iostream>

using namespace std;

int function (int i,int *j) {
  i++;
  (*j)++;
  return i+*j; }

int main () {
  int a=3,b=7;
  cout<<a<<" "<<b<<endl;
  cout<<function (a,&b)<<endl;
  cout<<a<<" "<<b<<endl;
  cout<<function (a,&b)<<endl;
  cout<<a<<" "<<b<<endl;
  return 0; }
```

Wykonanie tego programu daje taki wydruk:

```
3 7
12
3 8
13
3 9
```

2.5.4 Argument - pytanie

Co zostanie wypisane na ekranie w wyniku wykonania poniższego programu?

```
#include <iostream>

using namespace std;
```

```
int function (int a,int *b) {
  return *++b=a; }

int main () {
  int v [2]={1,2};
  cout<<(v [0]=function (v [1],v))<<" ";
  cout<<v [1]<<endl;
  return 0; }
```

2.5.5 Bubble

Sortowanie bąbelkowe odbywa się następująco. Przypuśćmy, że chcemy posortować wektor w kolejności rosnącej. Porównujemy najpierw dwa pierwsze elementy i jeśli są w złej kolejności, to zamieniamy. Następnie robimy to samo z drugim i trzecim elementem i tak dalej, do końca wektora. Jeżeli w takim pojedynczym przebiegu wszystkie pary były w dobrej kolejności to znaczy, że wektor jest już posortowany. Jeśli natomiast musieliśmy wykonać przynajmniej jedną zamianę, to powtarzmy całą procedurę od początku.

Napisz procedurę `bubble` sortującą wektor bąbelkowo w kolejności rosnącej. Deklaracja:

```
void bubble (double vec [],int n);
```

Argument	**Wejście**	**Wyjście**
`vec`	Wektor do posortowania	Wektor posortowany
`n`	Długość wektora	

Napisz program `bubble`, który wczytuje ze standardowego wejścia długość wektora, tworzy go w pamięci i wypełnia losowymi liczbami, wypisuje na standardowe wyjście, a następnie sortuje bąbelkowo i wypisuje wektor posortowany.

Pamiętaj, że sortowanie bąbelkowe jest najgorszym znanym algorytmem sortowania. Jeśli chcesz poznać lepsze algorytmy, wybierz studia na Wydziale Fizyki Uniwersytetu Warszawskiego.

```
#include <cstdlib>
#include <ctime>
#include <iostream>

using namespace std;

void bubble (double vec [],int n) {
  bool flag;
  do {
    flag=false;
    for (int i=0;i<n-1;i++) {
      if (vec [i+1]<vec [i]) {
        double tmp=vec [i];
        vec [i]=vec [i+1];
        vec [i+1]=tmp;
        flag=true; }}}
  while (flag); }

int main () {
```

```
  srandom (time (0));
  int n;
  cin>>n;
  double vec [n];
  for (int i=0;i<n;i++)
    vec [i]=random ()/RAND_MAX;
  for (int i=0;i<n;i++)
    cout<<vec [i]<<endl;
  cout<<endl;
  bubble (vec,n);
  for (int i=0;i<n;i++)
    cout<<vec [i]<<endl;
  return 0; }
```

2.5.6 Selection

Sortowanie metodą wybierania odbywa się następująco. Przypuśćmy, że chcemy posortować wektor w kolejności rosnącej. Wybieramy najpierw z całego wektora element najmniejszy i wstawiamy go na początek, tzn. zamieniamy miejscami z pierwszym. Następnie powtarzamy tę procedurą dla wektora bez pierwszego elementu itd.

Napisz procedurę `selection`, która sortuje wektor rosnąco metodą wybierania. Deklaracja:

```
void selection (double vec [],int n);
```

Argument	**Wejście**	**Wyjście**
`vec`	Wektor do posortowania	Wektor posortowany
`n`	Długość wektora	

Napisz program `selection`, który wczytuje ze standardowego wejścia długość wektora, tworzy go w pamięci i wypełnia losowymi liczbami, wypisuje na standardowe wyjście, a następnie sortuje metodą wybierania i wypisuje wektor posortowany.

```
#include <cstdlib>
#include <ctime>
#include <iostream>

using namespace std;

void selection (double vec [],int n) {
  for (int i=0;i<n-1;i++) {
    int m=i;
    for (int j=i+1;j<n;j++)
      if (vec [j]<vec [m])
        m=j;
    double t=vec [i];
    vec [i]=vec [m];
    vec [m]=t; }}

int main () {
  srandom (time (0));
```

```
  int n;
  cin>>n;
  double vec [n];
  for (int i=0;i<n;i++)
    vec [i]=random ()/RAND_MAX;
  for (int i=0;i<n;i++)
    cout<<vec [i]<<endl;
  cout<<endl;
  selection (vec,n);
  for (int i=0;i<n;i++)
    cout<<vec [i]<<endl;
  return 0; }
```

2.5.7 Insertion

Sortowanie metodą wstawiania odbywa się następująco. Przypuśćmy, że chcemy posortować wektor w kolejności rosnącej i że pierwszych $i-1$ elementów jest już posortowanych. Wyjmujemy element i-ty i wstawiamy go na właściwe miejsce pośród pierwszych i elementów, przesuwając wszystkie większe od niego o jeden w kierunku większych indeksów. W ten sposób posortowanych jest już pierwszych i elementów, co pozwala na powtórzenie procedury dla kolejnego elementu itd.

Napisz procedurę `insertion` sortującą rosnąco wektor `vec` o długości `n` metodą wstawiania. Deklaracja:

```
void insertion (double vec [],int n);
```

Argument	**Wejście**	**Wyjście**
`vec`	Wektor do posortowania	Wektor posortowany
`n`	Długość wektora	

Napisz program `insertion`, który wczytuje ze standardowego wejścia długość wektora, tworzy go w pamięci i wypełnia losowymi liczbami, wypisuje na standardowe wyjście, a następnie sortuje metodą wstawiania i wypisuje wektor posortowany.

```
#include <cstdlib>
#include <ctime>
#include <iostream>

using namespace std;

void insertion (double vec [],int n) {
  for (int i=1;i<n;i++) {
    double tmp=vec [i];
    int j;
    for (j=i-1;j>=0;j--) {
      if (vec [j]<=tmp)
        break;
      vec [j+1]=vec [j]; }
    vec [j+1]=tmp; }}

int main () {
```

```
  srandom (time (0));
  int n;
  cin>>n;
  double vec [n];
  for (int i=0;i<n;i++)
    vec [i]=random ()/RAND_MAX;
  for (int i=0;i<n;i++)
    cout<<vec [i]<<endl;
  cout<<endl;
  insertion (vec,n);
  for (int i=0;i<n;i++)
    cout<<vec [i]<<endl;
  return 0; }
```

2.5.8 Length

W języku `C++` łańcuchy, czyli zmienne tekstowe, reprezentujemy zazwyczaj przy pomocy typu `string`. Można jednak postępować jak w języku `C`, gdzie używa się do tego celu zwykłych tablic znaków. Znaki są to zmienne typu `char`, które zajmują jeden bajt i mogę być też traktowane jako liczby całkowite. Łańcuch może być krótszy niż zawierająca go tablica. Aby zaznaczyć, gdzie łańcuch się kończy, za jego ostatnim znakiem umieszcza się bajt zerowy. Oznacza to w szczególności, że długość tablicy musi być przynajmniej o jeden większa od liczby znaków zawartego w niej łańcucha, aby w jej ostatnim elemencie zmieścił się bajt zerowy.

Napisz funkcję `length` znajdującą długość łańcucha zapisanego w tablicy znaków, czyli liczbę znaków przed pierwszym bajtem zerowym. Deklaracja:

```
int length (char string []);
```

Argument	**Wejście**	**Wyjście**
`string`	Łańcuch	Niezmienione
`length`		Długość łańcucha

Napisz program `length` znajdujący i wypisujący na standardowe wyjście długość łańcucha `Vivat scientia physica!`.

```
#include <iostream>

using namespace std;

int length (char string []) {
  int i;
  for (i=0;string [i];i++);
  return i; }

int main () {
  char string [30]="Vivat scientia physica!";
  cout<<length (string)<<endl;
  return 0; }
```

Wykonanie tego programu wypisze na standardowe wyjście liczbę 23.

2.5.9 Spaces

Napisz funkcję `spaces`, która z łańcucha zapisanego w zadanej tablicy usuwa znajdujące się na końcu spacje przez wstawienie bajtu zerowego zaraz za ostatnim znakiem niebędącym spacją. Deklaracja:

```
void spaces (char string []);
```

Argument	**Wejście**	**Wyjście**
`string`	Tablica zawierająca łańcuch ze spacjami na końcu	Tablica zawierająca łańcuch bez spacji na końcu

W jednym pliku umieść funkcję `spaces` oraz następujący program główny:

```
int main () {
  char string [30]="Vivat scientia physica!   ";
  cout<<string<<'*'<<endl;
  spaces (string);
  cout<<string<<'*'<<endl;
  return 0; }
```

Wynikiem wykonania tego programu powinno być:

```
Vivat scientia physica!   *
Vivat scientia physica!*
```

```
#include <iostream>

using namespace std;

void spaces (char string []) {
  int i;
  for (i=0;string [i];i++);
  for (;i>0 && string [i-1]==' ';i--);
  string [i]=0; }

int main () {
  char string [30]="Vivat scientia physica!   ";
  cout<<string<<'*'<<endl;
  spaces (string);
  cout<<string<<'*'<<endl;
  return 0; }
```

2.5.10 Compare

Napisz funkcję `compare` określającą kolejność alfabetyczną dwóch zadanych łańcuchów znakowych. Deklaracja:

```
int compare (char first [],char second []);
```

Funkcja powinna zwracać 1 jeżeli łańcuch `first` wypada w kolejności alfabetycznej przed łańcuchem `second`, -1 w przeciwnym wypadku, oraz 0 jeżeli łańcuchy są identyczne. Jeżeli początek dłuższego łańcucha jest identyczny z krótszym łańcuchem, to pierwszy w kolejności alfabetycznej jest łańcuch krótszy. Łańcuchy mogą zawierać dowolne znaki ASCII, czyli duże i małe litery, cyfry, znaki interpunkcyjne itd. Przyjmij, że kolejność alfabetyczna znaków jest taka, jak kolejność ich kodów w systemie ASCII.

```
int compare (const char first [],const char second []) {
  int i;
  for (i=0;first [i] && first [i]==second [i];i++);
  return first [i]==second [i] ? 0 : first [i]<second [i] ? 1 : -1; }
```

2.5.11 Statistics

Niech dana będzie próba losowa N wartości x_i. Średnia i odchylenie standardowe z tej próby wynoszą odpowiednio

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2}$$

Napisz procedurę `statistics`, która oblicza średnią i odchylenie standardowe z próby. Deklaracja:

```
void statistics (double *s,double *m,double xx [],int n);
```

Argument	Wejście	Wyjście
`s`		σ_x
`m`		μ_x
`xx`	x_i	
`n`	N	

Napisz program `statistics`, który pobiera próbę 1000 liczb pseudolosowych z rozkładu płaskiego w przedziale od 0 do 1, a następnie wypisuje na standardowe wyjście wartość średnią i odchylenie standardowe.

```
#include <iostream>
#include <cstdlib>
#include <cmath>

using namespace std;

void statistics (double *s,double *m,double xx [],int n) {
  *s=*m=0;
  for (int i=0;i<n;i++)
    *m+=xx [i];
  *m/=n;
  for (int i=0;i<n;i++)
    *s+=(xx [i]-*m)*(xx [i]-*m);
  *s=sqrt (*s/(n-1)); }
```

```
int main () {
  double xx [1000];
  for (int i=0;i<1000;i++)
    xx [i]=(double) random ()/RAND_MAX;
  double s,m;
  statistics (&s,&m,xx,1000);
  cout<<m<<" "<<s<<endl;
  return 0; }
```

2.5.12 Bisection

Napisz procedurę `bisection` znajdującą metodą bisekcji miejsce zerowe x_0 funkcji f leżące w zadanym przedziale (x_1, x_2). Wynik powinien być znajdowany z zadaną dokładnością ϵ. Deklaracja:

```
double bisection (double (*fun) (double),double x1,double x2,double eps);
```

Argument	Wejście	Wyjście
`fun`	f	
`x1`	x_1	
`x2`	x_2	
`eps`	ϵ	
`bisection`		x_0

Napisz program `bisection` rozwiązujący równanie $e^{-x} = x$ z dokładnością 1e-10 i wypisujący wynik na standardowe wyjście.

```
#include <cmath>
#include <iostream>

using namespace std;

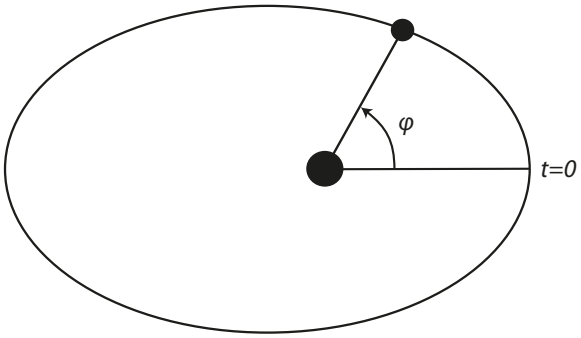
double bisection (double (*fun)(double),double x1,double x2,double eps) {
  double y1=fun (x1);
  while (fabs (x2-x1)>eps) {
    double x=(x1+x2)/2;
    double y=fun (x);
    if (y*y1>0) {
      x1=x;
      y1=y; }
    else
      x2=x; }
  return (x1+x2)/2; }

double fun (double x) {
  return exp (-x)-x; }

int main () {
  cout<<bisection (fun,0,1,1e-10)<<endl;
  return 0; }
```

2.5.13 Kepler

Rozważmy ruch ciała po elipsie w polu centralnej siły gratitacyjnej. Chcemy wyznaczyć położenie kątowe φ w funkcji czasu t. Kąt φ liczymy od peryhelium, gdzie ciało znajduje się w chwili $t = 0$, jak zaznaczono na rysunku.



Kąt φ wiąże się z tzw. anomalią mimośrodową u wzorami

$$\sin\varphi = \sqrt{1-\epsilon^2}\frac{\sin u}{1-\epsilon\cos u} \qquad \cos\varphi = \frac{\cos u - \epsilon}{1-\epsilon\cos u}$$

gdzie ϵ jest mimośrodem orbity. Anomalia mimośrodowa spełnia natomiast tzw. równanie Keplera,

$$u - \epsilon \in u = 2\pi\frac{t}{T}$$

gdzie T jest okresem obiegu. Widać, że położenie ciała zależy tylko od stosunku t/T. Dla danej wartości tego stosunku znajdujemy zatem anomalię mimośrodową rozwiązując równanie Keplera, a następnie przeliczmy ją na położenie kątowe przy pomocy podanych wzorów.

Napisz program `kepler` znajdujący położenie kątowe Ziemi φ w funkcji ułamka t/T. Mimośród orbity Ziemi wynosi 0.0167. Oblicz różnicę między φ a położeniem kątowym, jakie miałaby Ziemia w ruchu po orbicie kołowej z tym samym okresem. Sporządź wykres tej różnicy w funkcji t/T.

Obliczona tu różnica wiąże się z występującym w astronomii tzw. równaniem czasu. Oprócz eliptyczności orbity równanie to uwzględnia także nachylenie osi Ziemi.

```
#include <iostream>
#include <cmath>

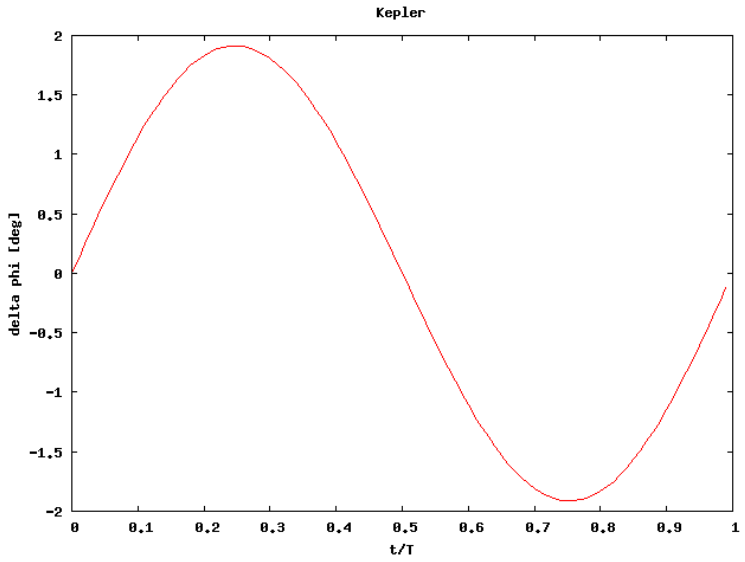
using namespace std;

const double e=0.0167;
double t;

double kepler (double u) {
  return u-e*sin (u)-2*M_PI*t; }

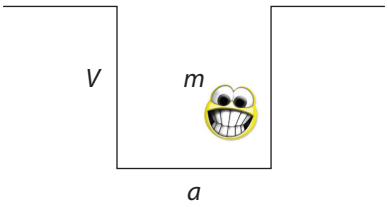
int main () {
  double u,f;
  for (t=0;t<1;t+=0.01) {
    u=bisection (kepler,0,2*M_PI);
```

```
  f=acos ((cos (u)-e)/(1-e*cos (u)));
  if (sin (u)<0)
    f=2*M_PI-f;
  cout<<t<<" "<<180*f/M_PI-360*t<<endl; }
return 0; }
```



2.5.14 Schrodinger

Każdy układ kwantowy posiada tak zwany stan podstawowy o pewnej energii. Rozważmy cząstkę o masie m znajdującą się w jednowymiarowej, prostokątnej studni potencjału o szerokości a i głębokości V, jak to pokazano na rysunku.



Okazuje się, że energia stanu podstawowego cząstki liczona względem dna studni dana jest wzorem

$$E = \frac{x^2}{r^2} V$$

gdzie

$$r = \frac{a}{2}\sqrt{\frac{2mV}{\hbar^2}}$$

zaś x jest najmniejszym dodatnim rozwiązaniem równania

$$\sqrt{r^2 - x^2} = x \tan x$$

Napisz program `schrodinger`, który wczytuje ze standardowego wejścia stałą r, a następnie wypisuje na standardowe wyjście energię stanu podstawowego liczoną w jednostkach głębokości studni, czyli wielkość x^2/r^2.

```
#include <iostream>
#include <cmath>

using namespace std;

double r;

double schrodinger (double x) {
  return sqrt (r*r-x*x)-x*tan (x); }

double bisection (double (*fun)(double),double x1,double x2,double eps) {
  double y1=fun (x1);
  do {
    double x=(x1+x2)/2;
    double y=fun (x);
    if (y*y1>0) {
      x1=x;
      y1=y; }
    else
      x2=x; }
  while (fabs (x2-x1)>2*eps);
  return (x1+x2)/2; }

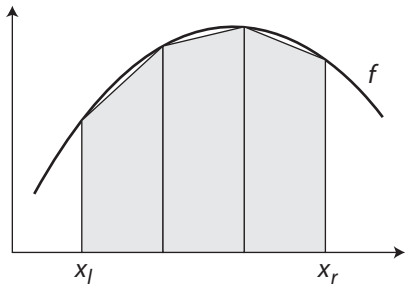
int main () {
  cin>>r;
  double x=bisection (schrodinger,0,r<M_PI/2 ? r : M_PI/2,1e-6);
  cout<<(x*x)/(r*r)<<endl;
  return 0; }
```

2.5.15 Trapezoid

Całkę

$$I = \int_a^b f(x)\mathrm{d}x$$

możemy przybliżyć sumą pól n trapezów o jednakowej szerokości zbudowanych pomiędzy osią odciętych a wykresem funkcji, jak to zaznaczono na rysunku dla $n = 3$.



Napisz funkcję `trapezoid`, która oblicza całkę metodą trapezów. Deklaracja:

```
double trapezoid (double (*f) (double),double a,double b,int n);
```

Argument	Wejście	Wyjście
`f`	f	
`a`	a	
`b`	b	
`n`	n	
`trapezoid`		I

Napisz program `trapezoid` obliczający całkę z funkcji sinus w przedziale od 0 do π z podziałem przedziału całkowania na 100 podprzedziałów.

```
#include <cmath>
#include <iostream>

using namespace std;

double trapezoid (double (*f) (double),double a,double b,int n) {
  double h=(b-a)/n,s=0;
  for (int i=0;i<n;i++)
    s+=h*(f (a+i*h)+f (a+(i+1)*h))/2;
  return s; }

int main () {
  cout<<trapezoid (sin,0,M_PI,100)<<endl;
  return 0; }
```

2.5.16 Ellipse

Napisz program `ellipse` obliczający obwód elipsy o zadanych półosiach. Program powinien wczytywać długości półosi ze standardowego wejścia, a następnie wypisywać wynik na standardowe wyjście. Całkowanie wykonaj we współrzędnych biegunowych metodą trapezów dzieląc przedział całkowania na 1000 podprzedziałów. We współrzędnych biegunowych elipsa o półosiach a i b ma równanie

$$r = \frac{ab}{\sqrt{a^2 \sin^2 \phi + b^2 \cos^2 \phi}}$$

```
#include <cmath>
#include <iostream>

using namespace std;

double trapezoid (double (*f) (double),double a,double b,int n) {
  double h=(b-a)/n,s=0;
  for (int i=0;i<n;i++)
    s+=h*(f (a+i*h)+f (a+(i+1)*h))/2;
  return s; }

double a,b;

double ellipse (double phi) {
```

```
  double s=sin (phi),c=cos (phi);
  return sqrt (a*a*b*b*pow (a*a*s*s+b*b*c*c,-1)
              +a*a*b*b*(b*b-a*a)*s*s*c*c*pow (a*a*s*s+b*b*c*c,-3)); }

int main () {
  cin>>a>>b;
  cout<<trapezoid (ellipse,0,2*M_PI,1000)<<endl;
  return 0; }
```

2.6 Struktury

2.6.1 Person - przykład

Poniższy przykład ilustruje sposób dostępu do pól struktury danej przez nazwę i wskaźnik oraz dynamiczne tworzenie obiektów.

```
#include <iostream>
#include <string>

using namespace std;

struct Person {
  string name;
  int birth; };

int main () {
  Person student={"Albert",1879};
  cout<<student.name<<" "<<student.birth<<endl;
  student.name="Einstein";
  Person *doctor=&student;
  cout<<doctor->name<<" "<<doctor->birth<<endl;
  doctor=new Person;
  doctor->name="Oetker";
  doctor->birth=1862;
  cout<<doctor->name<<" "<<doctor->birth<<endl;
  delete doctor;
  return 0; }
```

Wykonanie tego programu daje następujący wynik:

```
Albert 1879
Einstein 1879
Oetker 1862
```

2.6.2 Reversi

Zaimplementuj stos zmiennych typu `string`. Korzystając z tej struktury napisz program `reversi`, który wczytuje ze standardowego wejścia ciąg słów, a następnie wypisuje je na standardowe wyjście w odwrotnej kolejności, oddzielone spacjami. Program przetestuj na tekście *Pana Tadeusza* i *Hamleta*.

```
#include <iostream>
#include <string>

using namespace std;

struct Item {
  string value;
  Item *lower; };

Item *top=0;

void push (string value) {
  Item *auxiliary=top;
  top=new Item ();
  top->value=value;
  top->lower=auxiliary; }

string pop () {
  string value=top->value;
  Item *auxiliary=top->lower;
  delete top;
  top=auxiliary;
  return value; }

int main () {
  string word;
  while (cin>>word)
    push (word);
  while (top)
    cout<<pop ()<<" ";
  cout<<endl;
  return 0; }
```

2.6.3 System

Do przedstawienia zadanej liczby całkowitej nieujemnej n w systemie pozycyjnym o podstawie m można posłużyć się stosem liczb całkowitych. Algorytm przedstawia się wtedy następująco. Resztę z dzielenia n przez m odkładamy na stos, a następnie zastępujemy n ilorazem z dzielenia n przez m. Czynności te powtarzamy dopóki n jest niezerowe. Następnie kolejno zdejmujemy liczby ze stosu i wypisujemy od lewej do prawej z tym, że zamiast liczby `10` wypisujemy literę `A` i tak dalej.

Napisz program `system`, który wczytuje ze standardowego wejścia liczby n oraz m, a następnie wypisuje na standardowe wyjście liczbę n w sytemie pozycyjnym o podstawie m. Program powinien zawierać własną implementację stosu liczb całkowitych.

```
#include <iostream>

using namespace std;

struct Item {
```

```
  int value;
  Item *lower; };

Item *top=0;

void push (int value) {
  Item *auxiliary=top;
  top=new Item ();
  top->value=value;
  top->lower=auxiliary; }

int pop () {
  int value=top->value;
  Item *auxiliary=top->lower;
  delete top;
  top=auxiliary;
  return value; }

int main () {
  int number,system;
  cin>>number>>system;
  do {
    push (number%system);
    number/=system; }
  while (number);
  while (top) {
    int digit=pop ();
    cout<<(digit<10 ? (char) ('0'+digit) : (char) ('A'+digit-10)); }
  cout<<endl;
  return 0; }
```

2.7 Referencje

2.7.1 Reference - przykład

Poniższy program ilustruje wykorzystanie zmiennych referencyjnych.

```
#include <iostream>

using namespace std;

int main () {
  int a=5,&r=a;
  cout<<a<<' '<<r<<endl;
  a=7;
  cout<<a<<' '<<r<<endl;
  r=9;
  cout<<a<<' '<<r<<endl;
  return 0; }
```

Jego wykonanie daje taki wynik:

```
5 5
7 7
9 9
```

2.7.2 Function - przykład

Poniższy program ilustruje wykorzystanie referencji przy przekazywaniu argumentów i zwracaniu wartości przez funkcję.

```
#include <iostream>

using namespace std;

int &function (int &r) {
  return ++r; }

int main () {
  int a=7;
  cout<<a<<endl;
  cout<<function (a)<<endl;
  cout<<a<<endl;
  cout<<(function (a)=3)<<endl;
  cout<<a<<endl;
  return 0; }
```

Jego wykonanie daje taki wynik:

```
7
8
8
3
3
```

2.7.3 Max

Napisz funkcję `max`, która zwraca większą z dwóch podanych zmiennych całkowitych oraz umożliwia nadanie jej nowej wartości. Funkcji powinno się dać użyć następująco:

```
  int a=3,b=7;
  max (a,b)=0;
  cout<<a<<" "<<b<<endl;
```

Wynikiem działania tego fragmentu programu powinno być wypisanie liczb `3 0`.

```
#include <iostream>

using namespace std;

int &max (int &x,int &y) {
  return x>y ? x : y; }
```

```
int main () {
  int a=3,b=7;
  max (a,b)=0;
  cout<<a<<" "<<b<<endl;
  return 0; }
```

2.7.4 Characters

Napisz procedurę `characters`, która zlicza wystąpienia danego znaku w danym łańcuchu. Procedura powinna dać się zastosować w poniższym programie:

```
  string word="Trentatre Trentini entrarono in Trento
               tutti e trentatre trotterellando.";
  cout<<characters (word,'t')<<endl;
  cout<<characters ("Opasly susel spal sobie smacznie.",'s')<<endl;
```

którego wynikiem powinno być wypisanie liczb 14 i 6. Do procedury nie wolno przekazywać obiektów klasy `string`.

```
#include <iostream>
#include <string>

using namespace std;

int characters (const string &text,char c) {
  int n=0;
  for (int i=0;i<text.size ();i++)
    if (text [i]==c)
      n++;
  return n; }

int main () {
  string word="Trentatre Trentini entrarono in Trento
               tutti e trentatre trotterellando.";
  cout<<characters (word,'t')<<endl;
  cout<<characters ("Opasly susel spal sobie smacznie.",'s')<<endl;
  return 0; }
```

2.7.5 Capital

Napisz funkcję `capital`, która w danym łańcuchu zamienia wszystkie małe litery na wielkie. Funkcji powinno się dać użyć w następujący sposób:

```
  string s="TaK mOZna!";
  cout<<capital (s)<<endl;
  cout<<s<<endl;
```

Wynikiem działania tego fragmentu programu powinno być dwukrotne wypisanie napisu `TAK MOZNA!`. Funkcji nie powinno się dać użyć w następujący sposób:

```
  capital (s)="taK nIe moZnA!";
```

```
#include <iostream>
#include <string>

using namespace std;

const string &capital (string &text) {
  for (int i=0;i<text.size ();i++)
    if (text [i]>='a' && text [i]<='z')
      text [i]+='A'-'a';
  return text; }

int main () {
  string s="TaK mOZna!";
  cout<<capital (s)<<endl;
  cout<<s<<endl;
  return 0; }
```

2.7.6 Spacer

Strumień wejściowy zawiera słowa oddzielone różnymi liczbami znaków białych. Napisz procedurę `spacer`, która przepisuje te słowa do strumienia wyjściowego oddzielając je pojedynczymi spacjami. Procedurę powinno się dać zastosować w programie:

```
  istringstream si;
  ostringstream so;
  spacer (so,si);
  ifstream fi ("input.txt");
  ofstream fo ("output.txt");
  spacer (fo,fi);
  spacer (cout,cin);

#include <fstream>
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

void spacer (ostream &output,istream &input) {
  string word;
  if (input>>word)
    output<<word;
  while (input>>word)
    output<<" "<<word; }

int main () {
  istringstream si;
  ostringstream so;
  spacer (so,si);
  ifstream fi ("input.txt");
```

```
  ofstream fo ("output.txt");
  spacer (fo,fi);
  spacer (cout,cin);
  return 0; }
```

2.7.7 Words

Napisz procedurę `words`, która zlicza wystąpienia danego słowa w danym strumieniu wejściowym. Procedurę powinno się dać zastosować następująco:

```
  istringstream si;
  cout<<words (si,"veni")<<endl;
  ifstream fi;
  cout<<words (fi,"vidi")<<endl;
  cout<<words (cin,"vici")<<endl;
```

Z wykorzystaniem tej procedury napisz program `words`, który przyjmuje jako argumenty wywołania nazwę pliku oraz słowo, a następnie wypisuje na standardowe wyjście liczbę wystąpień tego słowa w pliku.

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

using namespace std;

int words (istream &stream,const string &key) {
  int i=0;
  string word;
  while (stream>>word)
    if (word==key)
      ++i;
  return i; }

int main (int argc,char *argv []) {
  ifstream file (argv [1]);
  cout<<words (file,argv [2])<<endl;
  file.close ();
  return 0; }
```

2.7.8 Temporary

Które z poniższych funkcji i programów są poprawne, a które błędne?

1. ```
   int &function (int a) {
     int b=a+3;
     return b; }
   ```

2. ```
   int function (int &a) {
     int b=a+3;
     return b; }
   ```

3.
```
int &function (int &r) {
  return ++r; }
```

4.
```
int &function (int &r) {
  return r++; }
```

5.
```
int function (int &r) {
  return r++; }
```

6.
```
void function (int &r) {
  cout<<r<<endl; }

int main () {
  function (7);
  return 0; }
```

7.
```
void function (const int &r) {
  cout<<r<<endl; }

int main () {
  function (7);
  return 0; }
```

8.
```
void function (const int &r) {
  cout<<r++<<endl; }
```

9.
```
void function (int &r) {
  cout<<r++<<endl; }

int main () {
  int a=25;
  function (a);
  return 0; }
```

10.
```
int &function (const int &r) {
  return r; }
```

11.
```
const int &function (const int &r) {
  return r; }

int main () {
  cout<<function (7)<<endl;
  return 0; }
```

12.
```
const int &function (int &r) {
  return r; }

int main () {
  int a=7;
  function (a)=5;
  return 0; }
```

13.
```
int &function (int &r) {
  return ++r; }

int main () {
  int a=7;
  function (a)=5;
  return 0; }
```

3 Łańcuchy, strumienie, pliki

3.1 Łańcuchy

3.1.1 Words

Napisz program `words` obliczający liczbę słów oraz średnią długość słowa w zadanym pliku tekstowym. Program powinien czytać plik ze standardowego wejścia, a wynik wypisywać na standardowe wyjście. Program przetestuj na tekście *Pana Tadeusza* i *Hamleta*.

```
#include <iostream>
#include <string>

using namespace std;

int main () {
  int length=0,number=0;
  for (string word;cin>>word;length+=word.size (),++number);
  cout<<number<<" "<<(double) length/number<<endl;
  return 0; }
```

3.1.2 Find

Napisz program `find` wypisujący na standardowe wyjście te linie ze standardowego wejścia, które zawierają zadane słowo podane jako argument wywołania programu.

```
#include <iostream>
#include <string>

using namespace std;

int main (int argc,char *argv []) {
  string line;
  while (getline (cin,line))
    if (linia.find (argv [1])!=string::npos)
      cout<<line<<endl;
  return 0; }
```

3.1.3 Palindrom

Napisz procedurę `palindrom` sprawdzającą, czy dany łańcuch typu `string` jest palindromem. Palindrom to napis, który czytany po literce od tyłu jest taki sam, jak czytany od przodu. Przy tym znaczenie mają tylko występujące w napisie litery, zaś pozostałe znaki, jak cyfry,

znaki interpunkcyjne i znaki białe, są ignorowane. Poza tym nie ma znaczenia wielkość liter. Na przykład palindromem jest zdanie *Ile Roman ładny dyndał na moreli.*

Korzystając z tej procedury napisz program `palindrom`, który wczytuje ze standardowego wejścia jedną linię, a następnie wypisuje na standardowe wyjście komunikat, czy zawarty w tej linii tekst jest palindromem.

```
#include <iostream>
#include <string>

using namespace std;

bool palindrom (const string &text) {
  string letters;
  for (int i=0;i!=text.size ();++i) {
    if ('a'<=text [i] && text [i]<='z')
      letters+=text [i];
    if ('A'<=text [i] && text [i]<='Z')
      letters+=text [i]+('a'-'A'); }
  for (int i=0;i!=letters.size ();++i)
    if (letters [i]!=letters [letters.size ()-i-1])
      return false;
  return true; }

int main () {
  string text;
  getline (cin,text);
  cout<<(palindrom (text) ? "Yes" : "No")<<endl;
  return 0; }
```

3.2 Strumienie

3.2.1 Letters

Napisz program `letters` liczący wystąpienia każdej litery alfabetu w zadanym pliku tekstowym. Nie uwzględniaj wielkości liter ani znaków nie będących literami. Wynikiem powinna być tabela zawierająca w jednej kolumnie litery, a w drugiej liczby wystąpień. Dane należy czytać ze standardowego wejścia, a wynik wypisywać na standardowe wyjście. Program przetestuj na tekście *Pana Tadeusza* i *Hamleta.*

```
#include <iostream>

using namespace std;

int main () {
  int spectrum [26];
  for (int index=0;index<26;index++)
    spectrum [index]=0;
  char letter;
  while (cin>>letter)
    if (letter>='a' && letter<='z')
      spectrum [letter-'a']++;
```

```
    else if (letter>='A' && letter<='Z')
      spectrum [letter-'A']++;
  for (int index=0;index<26;index++)
    cout<<(char) (index+'a')<<' '<<spectrum [index]<<endl;
  return 0; }
```

3.2.2 Enigma

W plikach tekstowych każdy znak zapisany jest jako liczba jednobajtowa zwana kodem znaku. Jedną z metod szyfrowania wiadomości jest zastąpienie kodu każdego znaku jego bitową różnicą symetryczną (XOR) z zadaną jednobajtową liczbą zwaną kluczem. Aby odczytać wiadomość, wystarczy ponownie wziąć bitową różnicę symetryczną zaszyfrowanych kodów z tym samym kluczem.

Umawiamy się, że zaszyfrowane wiadomości będziemy zapisywać przy pomocy liczb oddzielonych spacjami, np. `17 145 2 8`. Szyfrowanie odbywa się znak po znaku, włączając spacje, tabulatory, znaki końca linii itp.

Napisz program `enigma` szyfrujący i rozszyfrowujący wiadomości tekstowe przy pomocy różnicy symetrycznej z kluczem. Program powinien przyjmować dwa argumenty wywołania. Pierwszy to klucz, np. `17`, a drugi to `e` lub `d` odpowiednio dla szyfrowania i rozszyfrowywania. Program powinien czytać ze standardowego wejścia i pisać na standardowe wyjście.

Przetestuj program zapisując wiadomość tekstową np. w pliku `enigma.txt`, a następnie szyfrując ją i rozszyfrowując z tym samym kluczem, np. `17`, przy pomocy polecenia

```
./enigma 17 e <enigma.txt | ./enigma 17 d
```

```
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

int main (int argc,char *argv []) {
  int key;
  istringstream (argv [1])>>key;
  if ((string) argv [2]=="e") {
    char letter;
    while (cin.get (letter))
      cout<<(letter^key)<<' '; }
  else {
    int number;
    while (cin>>number)
      cout<<(char) (number^key); }
  return 0; }
```

3.3 Strumienie związane z łańcuchami

3.3.1 Column

Napisz program `column` wypisujący na standardowe wyjście n-tą kolumnę pliku czytanego ze standardowego wejścia z zachowaniem podziału na linie. Kolumny są to ciągi znaków oddzielone spacjami lub tabulatorami. Jeżeli dana linia zawiera mniej niż n kolumn, to wypisywana powinna być linia pusta. Numer kolumny n powinien być podawany jako argument wywołania programu.

```
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

int main (int argc,char *argv []) {
  int number;
  istringstream (argv [1])>>number;
  string line;
  while (getline (cin,line)) {
    istringstream stream (line);
    string word;
    for (int counter=0;counter!=number;++counter)
      if (!(stream>>word)) {
        word.clear ();
        break; }
    cout<<word<<endl; }
  return 0; }
```

3.3.2 Cinderella

Standardowy strumień wejściowy zawiera wyłącznie liczby naturalne i słowa złożone z małych liter alfabetu łacińskiego. Słowa i liczby są przemieszane między sobą i oddzielone pojedynczymi spacjami. Napisz program `cinderella`, który odczytuje całą zawartość standardowego wejścia i wypisuje na standardowe wyjście sumę wszystkich liczb oraz całkowitą liczbę liter we wszystkich słowach.

```
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

int main () {
  int sum=0,letters=0;
  string item;
  while (cin>>item) {
    istringstream stream (item);
    int number;
    if (stream>>number)
      sum+=number;
    else
      letters+=item.size (); }
  cout<<"Sum of numbers: "<<sum<<endl;
  cout<<"Number of letters: "<<letters<<endl;
  return 0; }
```

3.3.3 Invoice

Począwszy od bieżącego semestru studentom zarejestrowanym w USOS nasz wydziałowy barek sprzedaje dania na kredyt. Następnie przesyła u-mailem plik z rachunkiem w formacie takim, jak w poniższym przykładzie:

```
sałata księżycowa 2.50
czarodziejski filet 7.20
2 konserwy brandenburskie 6.33
napój energetyczny 10.5 13.70
```

Cena jest ostatnią pozycją w linii.

Napisz program `invoice` obliczający na podstawie tego pliku całkowitą należność. Plik z rachunkiem powinien być wczytywany ze standardowego wejścia, a wynik wypisywany na standardowe wyjście.

```
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

int main () {
  double sum=0;
  string line;
  while (getline (cin,line)) {
    double price;
    string word;
    for (istringstream stream (line);stream>>word;);
    istringstream stream (word);
    if (stream>>price)
      sum+=price; }
  cout<<sum<<endl;
  return 0; }
```

3.4 Pliki

3.4.1 Zero

Pewien plik tekstowy ma następującą strukturę:

```
      Time       Mass     Volume     Charge   Velocity
 1.677e+01  0.000e+00  0.000e+00  0.000e+00  4.237e+00
 2.223e+01  0.000e+00  0.000e+00  0.000e+00 -5.503e+00
 2.947e+01  0.000e+00  1.767e-03  0.000e+00  7.067e+00
 3.907e+01  3.073e-02  1.767e-03  0.000e+00 -9.027e+00
```

Liczba wierszy, liczba kolumn, ani nagłówki kolumn nie są z góry znane. Wiadomo jednak, że każda kolumna ma swój nagłówek składający się z jednego słowa zawierającego przynajmniej jeden i najwyżej 10 znaków. Wszystkie liczby zapisane są w formacie naukowym z czterema cyframi znaczącymi. Kolumny mają szerokość 10 znaków i są oddzielone pojedynczymi spacjami.

Napisz program `zero`, który usuwa z takiego pliku kolumny zawierające same zera wraz z ich nagłówkami. Plik wyjściowy powinien być zapisywany w takim samym formacie, jak plik wejściowy. Przed pierwszą ani za ostatnią kolumną nie powinny być wypisywane zbędne spacje. W podanym przykładzie wynik działania programu powinien być następujący:

```
      Time       Mass     Volume   Velocity
 1.677e+01  0.000e+00  0.000e+00  4.237e+00
 2.223e+01  0.000e+00  0.000e+00 -5.503e+00
 2.947e+01  0.000e+00  1.767e-03  7.067e+00
 3.907e+01  3.073e-02  1.767e-03 -9.027e+00
```

Nazwa pliku powinna być podawana jako argument wywołania programu.

```
#include <fstream>
#include <iomanip>
#include <sstream>
#include <string>
#include <vector>

using namespace std;

int main (int argc,char *argv []) {
  ifstream input (argv [1]);
  string line;
  vector <string> headers;
  getline (input,line);
  istringstream stream (line);
  string header;
  while (stream>>header)
    headers.push_back (header);
  vector <vector <double> > rows;
  while (getline (input,line) && line.size ()) {
    vector <double> row (headers.size ());
    istringstream stream (line);
    for (int i=0;i!=row.size ();++i)
      stream>>row [i];
    rows.push_back (row); }
  input.close ();
  vector <bool> flags (headers.size (),false);
  for (int i=0;i!=rows.size ();++i)
    for (int j=0;j!=rows [i].size ();++j)
      if (rows [i][j])
        flags [j]=true;
  ofstream output (argv [1]);
  bool flag=false;
  for (int i=0;i!=headers.size ();++i)
    if (flags [i]) {
      if (flag)
        output<<" ";
      output<<setw (10)<<headers [i];
```

```
      flag=true; }
  output<<endl;
  output<<scientific<<setprecision (3);
  for (int i=0;i!=rows.size ();++i) {
    bool flag=false;
    for (int j=0;j!=rows [i].size ();++j)
      if (flags [j]) {
        if (flag)
          output<<" ";
        output<<setw (10)<<rows [i][j];
        flag=true; }
      output<<endl; }
  output.close ();
  return 0; }
```

3.4.2 Replace

Napisz program `replace`, który w każdej linii pliku tekstowego zastępuje wszystkie wystąpienia zadanego ciągu znaków innym ciągiem znaków. Nazwa pliku, słowo zastępowane oraz zastępujące powinny być kolejnymi argumentami wywołania programu.

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdio>

using namespace std;

int main (int argc,char *argv []) {
  ifstream input (argv [1]);
  char name [L_tmpnam];
  ofstream output (tmpnam (name));
  string line,previous=argv [2],next=argv [3];
  while (getline (input,line)) {
    for (int index=0;(index=line.find (previous,index))!=string::npos;) {
      line.replace (index,previous.size (),next);
      index+=next.size (); }
    output<<line<<endl; }
  output.close ();
  input.close ();
  rename (name,argv [1]);
  return 0; }
```

3.4.3 Spaces

Napisz program `spaces`, który w zadanym pliku tekstowym usuwa białe znaki znajdujące się na końcach wierszów. Nazwa pliku powinna być argumentem wywołania programu.

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

```
#include <string>
#include <cstdio>
#include <cctype>

using namespace std;

int main (int argc,char *argv []) {
  ifstream input (argv [1]);
  char name [L_tmpnam];
  ofstream output (tmpnam (name));
  string line;
  while (getline (input,line)) {
    int index;
    for (index=line.size ();index>0 && isspace (line [index-1]);index--);
    line.erase (index);
    output<<line<<endl; }
  output.close ();
  input.close ();
  rename (name,argv [1]);
  return 0; }
```

3.4.4 Liner

Plik tekstowy zawiera słowa oddzielone znakami białymi. Napisz program `liner`, który przeformatuje ten plik oddzielając słowa pojedynczymi spacjami i dzieląc tekst na linie bez łamania słów. Każda linia powinna zawierać największą możliwą liczbę słów tak, aby nie została przekroczona dopuszczalna długość linii, rozumiana jako liczba występujących w niej znaków, łącznie ze spacjami. Nazwa pliku do przeformatowania oraz dopuszczalna długość linii powinny być zadawane jako argumenty wywołania. Program przetestuj na tekście *Pana Tadeusza* i *Hamleta*.

```
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <cstdio>

using namespace std;

int main (int argc,char *argv []) {
  ifstream input (argv [1]);
  char name [L_tmpnam];
  ofstream output (tmpnam (name));
  int max;
  istringstream (argv [2])>>max;
  int length=0;
  string word;
  while (input>>word) {
    if (length+word.size ()+1<=max) {
      if (length)
        output<<' ';
```

```
      output<<word;
      length+=word.size ()+1; }
    else {
      output<<"\n"<<word;
      length=word.size (); }}
  output<<"\n";
  output.close ();
  input.close ();
  rename (name,argv [1]);
  return 0; }
```

3.4.5 Nucleus

Fizyk jądrowy ściągnął z internetu plik

http://amdc.in2p3.fr/masstables/Ame2011int/mass.mas114

zawierający najnowsze dane o masach jąder atomowych. W pracy nad doktoratem będzie z niego odczytywał energie wiązania na nukleon (`BINDING ENERGY/A`) dla różnych jąder. Fizyk stwierdził jednak, że struktura pliku nie jest przyjazna użytkownikowi i postanowił napisać program, który wczytuje ze standardowego wejścia liczbę atomową `Z` oraz liczbę masową `A` i na podstawie pliku wypisuje na standardowe wyjście odpowiadającą im energię wiązania na nukleon. Napisz program `nucleus`, który realizuje to zadanie. Program powinien odczytywać wartości `Z` i `A` do napotkania końca pliku wypisując energię wiązania na nukleon dla każdej odczytanej pary.

```
#include <fstream>
#include <iostream>
#include <map>
#include <sstream>
#include <string>

using namespace std;

int main () {
  map <pair <int,int>,double> energies;
  ifstream file ("mass.mas114");
  string line;
  for (int i=0;i!=39;++i)
    getline (file,line);
  while (getline (file,line) && line.size ()) {
    int Z,A;
    double energy;
    istringstream (line.substr (9))>>Z;
    istringstream (line.substr (14))>>A;
    istringstream (line.substr (52))>>energy;
    energies [pair <int,int> (Z,A)]=energy; }
  file.close ();
  int Z,A;
  while (cin>>Z>>A)
```

```
    cout<<energies [pair <int,int> (Z,A)]<<endl;
  return 0; }
```

4 Obiektowość i dziedziczenie

4.1 Obiekty

4.1.1 Person - przykład

Niniejszy przykład ilustruje użycie funkcji składowych.

```
#include <iostream>
#include <string>

using namespace std;

struct Person {
  string name;
  int birth;
  void print (ostream &stream); };

void Person::print (ostream &stream) {
  stream<<name<<" "<<birth<<endl; }

int main () {
  Person student={"Albert",1879};
  student.name="Einstein";
  student.print (cout);
  Person *doctor=new Person2;
  doctor->name="Oetker";
  doctor->birth=1862;
  doctor->print (cout);
  delete doctor;
  return 0; }
```

Wykonanie tego programu daje następujący wynik:

```
Einstein 1879
Oetker 1862
```

4.1.2 Mass

Umawiamy się, że jeżeli masa jakiegoś ciała jest mniejsza od `1kg`, to zapisujemy ją w gramach, na przykład `23g`, jeżeli zawiera się miedzy `1kg` a `1T`, to zapisujemy ją w kilogramach, na przykład `78.3kg`, a jeżeli jest większa, to zapisujemy ją w tonach, na przykład `12T`, pisząc symbol jednostki bezpośrednio za wartością liczbową.

Napisz klasę `Mass` reprezentującą masę dowolnego ciała. Zaimplementuj:

- Konstruktor bezargumentowy inicjalizujący masę wartością zero.
- Konstruktor o jednym argumencie rzeczywistym, inicjalizujący masę wartością argumentu przyjmując, że jest ona podana w kilogramach.

- Konstruktor o jednym argumencie tekstowym zawierającym oznaczenie masy zgodne z podanym opisem, na przykład `1.5g` lub `17T`. Konstuktor powinien inicjalizować masę zadaną w ten sposób wartością.
- Operator konwersji do typu rzeczywistego. Wynikiem konwersji powinna być masa wyrażona w kilogramach.
- Operator konwersji do typu `string` zwracający oznaczenie masy zgodne z podanym opisem, na przykład `1.5g` lub `17T`.
- Operator porównania `<` i operator dodawania `+`.
- Operator `<<` wypisujący masę do strumienia typu `ostream` oraz operator `>>` wczytujący masę ze strumienia typu `istream`. Operatory te powinny wypisywać i wczytywać masę w formacie opisanym wyżej.

```
#include <istream>
#include <ostream>
#include <sstream>

using namespace std;

class Mass {
  double mass;
public:
  Mass ();
  Mass (double mass);
  Mass (const string &mass);
  operator double ();
  operator string ();
  friend bool operator < (const Mass &first,const Mass &second);
  friend Mass operator + (const Mass &first,const Mass &second);
  friend ostream &operator << (ostream &stream,const Mass &mass);
  friend istream &operator >> (istream &stream,Mass &mass); };

Mass::Mass (): mass () {}

Mass::Mass (double mass): mass (mass) {}

Mass::Mass (const string &mass) {
  istringstream stream (mass);
  stream>>*this; }

Mass::operator double () {
  return mass; }

Mass::operator string () {
  ostringstream stream;
  stream<<*this;
  return stream.str (); }
```

```
bool operator < (const Mass &first,const Mass &second) {
  return first.mass<second.mass; }

Mass operator + (const Mass &first,const Mass &second) {
  return Mass (first.mass+second.mass); }

ostream &operator << (ostream &stream,const Mass &mass) {
  if (mass.mass<1)
    stream<<1000*mass.mass<<"g";
  else if (mass.mass<1000)
    stream<<mass.mass<<"kg";
  else
    stream<<mass.mass/1000<<"T";
  return stream; }

istream &operator >> (istream &stream,Mass &mass) {
  string unit;
  stream>>mass.mass>>unit;
  if (unit=="g")
    mass.mass/=1000;
  else if (unit=="T")
    mass.mass*=1000;
  return stream; }
```

4.1.3 Date

Napisz klasę `Date` reprezentującą datę, czyli pamiętającą dzień, miesiąc i rok w postaci liczb całkowitych. Zaimplementuj:

- Konstruktor bezargumentowy inicjalizujący dzień, miesiąc i rok wartościami zerowymi oraz konstruktor trójargumentowy inicjalizujący je swoimi argumentami.
- Operator preinkrementacji zwiększający datę o jeden dzień.
- Operator `<` odpowiadający kolejności chronologicznej dat.
- Operator `<<` wypisujący datę do strumienia typu `ostream` oraz operator `>>` wczytujący datę ze strumienia typu `istream`. Datę zapisujemy jak w przykładzie `31.12.2010`.

Napisz program `date`, który wczytuje ze standardowego wejścia dwie daty, a następnie wypisuje na standardowe wyjście daty 7 kolejnych dni następujących bezpośrednio po większej z podanych dat.

```
#include <iostream>

using namespace std;

class Date {
  static int lengths [12];
  int day,month,year;
public:
  Date ();
```

```
  Date (int day,int month,int year);
  Date &operator++ ();
  friend bool operator< (const Date &first,const Date &second);
  friend ostream &operator<< (ostream &stream,const Date &date);
  friend istream &operator>> (istream &stream,Date &date); };

int Date::lengths [12]={31,28,31,30,31,30,31,31,30,31,30,31};

Date::Date (): day (),month (),year () {}

Date::Date (int day,int month,int year): day (day),month (month),year (year) {}

Date &Date::operator++ () {
  lengths [1]=(year%4 ? 28 : 29);
  if (++day>lengths [month-1]) {
    day=1;
    if (++month>12) {
      month=1;
      ++year; }}
  return *this; }

bool operator< (const Date &first,const Date &second) {
  return first.year<second.year
     || (first.year==second.year && (first.month<second.month
                                  || (first.month==second.month && first.day<second.day))); }

ostream &operator<< (ostream &stream,const Date &date) {
  return stream<<date.day<<'.'<<date.month<<'.'<<date.year; }

istream &operator>> (istream &stream,Date &date) {
  char character;
  stream>>date.day>>character>>date.month>>character>>date.year; }

int main () {
  Date first,second;
  cin>>first>>second;
  for (int i=0;i<7;++i)
    cout<<++(first<second ? second : first)<<endl;
  return 0; }
```

4.1.4 Reversi

Napisz klasę `Stack` reprezentującą stos łańcuchów tekstowych. Zaimplementuj:

- Konstruktor bezargumentowy.
- Funkcję składową `push` odkładającą element na stos.
- Funkcję składową `pop` zdejmującą element ze stosu i zwracającą informację o tym, czy operacja się powiodła, to znaczy czy stos nie był pusty.

- Destruktor zwalniający całą pamięć używaną przez stos.

Napisz program `reversi`, który wczytuje ciąg słów, a następnie wypisuje je w odwrotnej kolejności, oddzielone spacjami. Program przetestuj na tekście *Pana Tadeusza* i *Hamleta*.

```
#include <iostream>
#include <string>

using namespace std;

class Stack {
  struct Item {
    string value;
    Item *lower; };
  Item *top;
public:
  Stack ();
  ~Stack ();
  void push (const string &value);
  bool pop (string &value); };

Stack::Stack (): top (0) {}

Stack::~Stack () {
   while (top) {
     Item *auxiliary=top->lower;
     delete top;
     top=auxiliary; }}

void Stack::push (const string &value) {
  Item *auxiliary=top;
  top=new Item ();
  top->value=value;
  top->lower=auxiliary; }

bool Stack::pop (string &value) {
  if (!top)
    return false;
  value=top->value;
  Item *auxiliary=top->lower;
  delete top;
  top=auxiliary;
  return true; }

int main () {
  string word;
  Stack stack;
  while (cin>>word)
    stack.push (word);
  while (stack.pop (word))
```

```
    cout<<word<<" ";
  cout<<endl;
  return 0; }
```

4.1.5 Triangle

Napisz klasę `Triangle` reprezentującą trójkąt o zadanych długościach boków. Zaimplementuj:

- Konstruktor trójargumentowy inicjalizujący długości boków swoimi argumentami.
- Funkcję składową `scale` przekształcającą trójkąt przez podobieństwo w skali zadanej swoim argumentem.
- Funkcję zaprzyjaźnioną `area` zwracającą pole trójkąta.

Napisz program `triangle`, który wczytuje długości boków trójkąta, wypisuje pole, a następnie wczytuje liczbę rzeczywistą i wypisuje pole wyjściowego trójkąta przekształconego przez podobieństwo w skali zadanej tą liczbą.

Wskazówka. Pole trójkąta S wyraża się przez długości boków a, b, c, wzorem Herona,

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

gdzie p jest połową obwodu.

```
#include <iostream>
#include <cmath>

using namespace std;

class Triangle {
  double a,b,c;
public:
  Triangle (double a,double b,double c);
  void scale (double s);
  friend double area (const Triangle &triangle); };

Triangle::Triangle (double a,double b,double c): a (a),b (b),c (c) {}

void Triangle::scale (double s) {
  a*=s;
  b*=s;
  c*=s; }

double area (const Triangle &triangle) {
  double p=(triangle.a+triangle.b+triangle.c)/2;
  return sqrt (p*(p-triangle.a)*(p-triangle.b)*(p-triangle.c)); }

int main () {
  double a,b,c;
  cin>>a>>b>>c;
  Triangle triangle (a,b,c);
  cout<<area (triangle)<<endl;
```

```
  double s;
  cin>>s;
  triangle.scale (s);
  cout<<area (triangle)<<endl;
  return 0; }
```

4.1.6 Name

Napisz klasę `Name`, której obiekt pamięta imię i nazwisko. Zaimplementuj:

- Konstruktor bezargumentowy inicjalizujący imię i nazwisko łańcuchami pustymi.
- Dwuargumentowy konstruktor inicjalizujący imię i nazwisko swoimi argumentami.
- Funkcję składową `initials` zwracającą łańcuch zawierający inicjały, np. `PO`. Przyjmij dla uproszczenia, że imiona i nazwiska są jednoczłonowe.
- Operator `<` wyznaczający kolejność alfabetyczną. Operator powinien porównywać najpierw nazwiska, a jeżeli są takie same, to imiona.
- Operator `<<` wypisujący imię i nazwisko do strumienia typu `ostream` oraz operator `>>` wczytujący je ze strumienia typu `istream`.

Napisz program `name`, który wczytuje ze standardowego wejścia dane dwóch osób, a następnie wypisuje na standardowe wyjście imię, nazwisko i inicjały osoby wypadającej wcześniej w kolejności alfabetycznej.

```
#include <iostream>
#include <string>

using namespace std;

class Name {
public:
  Name ();
  Name (const string &given,const string &family);
  string initials () const;
  friend bool operator< (const Name &first,const Name &second);
  friend ostream &operator<< (ostream &stream,const Name &Name);
  friend istream &operator>> (istream &stream,Name &Name);
private:
  string given,family; };

Name::Name (): given (),family () {}

Name::Name (const string &given,const string &family):
  given (given),family (family) {}

string Name::initials () const {
  return given.substr (0,1)+family.substr (0,1); }

bool operator< (const Name &first,const Name &second) {
```

```
  return first.family<second.family || (first.family==second.family && first.given<second.gi

ostream &operator<< (ostream &stream,const Name &Name) {
  return stream<<Name.given<<' '<<Name.family; }

istream &operator>> (istream &stream,Name &Name) {
  return stream>>Name.given>>Name.family; }

int main () {
  Name a,b;
  cin>>a>>b;
  if (a<b)
    cout<<a<<" "<<a.initials ()<<endl;
  else
    cout<<b<<" "<<b.initials ()<<endl;
  return 0; }
```

4.1.7 Velocity

Niech $\beta = v/c$ będzie prędkością wyrażoną w jednostkach prędkości światła. W przypadku jednowymiarowym jeżeli β_2 jest prędkością ciała 2 względem ciała 1, zaś β_1 jest prędkością ciała 1 w pewnym układzie odniesienia, to prędkość ciała 2 w tym układzie wynosi

$$\beta = \frac{\beta_1 + \beta_2}{1 + \beta_1\beta_2} \ .$$

Jest to relatywistyczne prawo składania prędkości.

Napisz klasę `Velocity` reprezentującą prędkość w ruchu jednowymiarowym. Zaimplementuj:

- Konstruktor, który można wywołać bez argumentów lub z jednym argumentem rzeczywistym. W pierwszym przypadku tworzony obiekt powinien być inicjalizowany wartością zero, a w drugim wartością podaną jako argument.
- Metodę `gamma` zwracającą wartość czynnika

$$\gamma = \frac{1}{\sqrt{1-\beta^2}}$$

- Operator `+=` zgodny z relatywistycznym prawem składania prędkości.
- Operator dodawania zgodny z relatywistycznym prawem składania prędkości.
- Operator `<<` wypisujący prędkość do strumienia typu `ostream` oraz operator `>>` wczytujący prędkość ze strumienia typu `istream`.

Napisz program `velocity`, który wczytuje dwie prędkości i wypisuje ich relatywistyczną sumę oraz odpowiadający jej czynnik γ.

```
#include <iostream>

using namespace std;

class Velocity {
```

```
public:
  Velocity (double value=0);
  double gamma ();
  friend Velocity operator+ (const Velocity &first,const Velocity &second);
  friend ostream &operator<< (ostream &stream,const Velocity &velocity);
  friend istream &operator>> (istream &stream,Velocity &velocity);
private:
  double value; };

Velocity::Velocity (double value): value (value) {}

double Velocity::gamma () {
  return 1/(1-value*value); }

Velocity operator+ (const Velocity &first,const Velocity &second) {
  return Velocity ((first.value+second.value)/(1+first.value*second.value)); }

ostream &operator<< (ostream &stream,const Velocity &velocity) {
  return stream<<velocity.value; }

istream &operator>> (istream &stream,Velocity &velocity) {
  return stream>>velocity.value; }

int main () {
  Velocity a,b;
  cin>>a>>b;
  cout<<a+b<<" "<<(a+b).gamma ()<<endl;
  return 0; }
```

4.1.8 Rest

Napisz klasę `Rest` reprezentującą liczby całkowite od 0 do 16 z dodawaniem i mnożeniem modulo 17. Zaimplementuj:

- Konstruktor, który można wywołać bez argumentów lub z jednym argumentem całkowitym. W pierwszym przypadku tworzony obiekt powinien być inicjalizowany wartością 0, a w drugim resztą z dzielenia podanej liczby przez 17.
- Operator preinkrementacji modulo 17.
- Operatory `+=` i `*=` działające modulo 17.
- Operatory dodawania i mnożenia modulo 17.
- Operator `<<` wypisujący do strumienia typu `ostream` liczbę reprezentowaną przez obiekt oraz operator `>>` wczytujący taką liczbę ze strumienia `istream`.

Napisz program `rest`, który wczytuje ze standardowego wejścia dwie liczby z przedziału od 0 do 16 i wypisuje na standardowe wyjście ich sumę oraz iloczyn modulo 17.

```
#include <iostream>
```

```
using namespace std;

class Rest {
public:
  Rest (int value=0);
  Rest &operator++ ();
  Rest &operator+= (const Rest &rest);
  Rest &operator*= (const Rest &rest);
  friend Rest operator+ (const Rest &first,const Rest &second);
  friend Rest operator* (const Rest &first,const Rest &second);
  friend ostream &operator<< (ostream &stream,const Rest &rest);
  friend istream &operator>> (istream &stream,Rest &rest);
private:
  int value; };

Rest::Rest (int value): value (value%17) {}

Rest &Rest::operator++ () {
  value=(value+1)%17;
  return *this; }

Rest &Rest::operator+= (const Rest &rest) {
  value=(value+rest.value)%17;
  return *this; }

Rest &Rest::operator*= (const Rest &rest) {
  value=(value*rest.value)%17;
  return *this; }

Rest operator+ (const Rest &first,const Rest &second) {
  return Rest (first.value+second.value); }

Rest operator* (const Rest &first,const Rest &second) {
  return Rest (first.value*second.value); }

ostream &operator<< (ostream &stream,const Rest &rest) {
  return stream<<rest.value; }

istream &operator>> (istream &stream,Rest &rest) {
  return stream>>rest.value; }

int main () {
  Rest a,b;
  cin>>a>>b;
  cout<<a+b<<" "<<a*b<<endl;
  return 0; }
```

4.1.9 Rat

Napisz klasę `Rat` reprezentującą liczby wymierne p/q. Liczby p i q powinny być pamiętane jako względnie pierwsze z q dodatnim. Zaimplementuj:

- Konstruktor, który można wywołać bez argumentów lub z jednym albo dwoma argumentami całkowitymi. W pierwszym przypadku tworzony obiekt powinien być inicjalizowany wartością zero, w drugim zadaną liczbą całkowitą, a w trzecim ilorazem dwóch argumentów.
- Funkcje składowe `numerator` i `denominator` zwracające odpowiednio licznik i mianownik liczby.
- Operator konwersji do typu `double`.
- Jednoargumentowy operator `-`.
- Operator `<`.
- Operator preinkrementacji.
- Operatory `+=` i `*=`.
- Operatory dodawania i mnożenia.
- Operator `<<` wypisujący reprezentowaną liczbę wymierną do strumienia typu `ostream` oraz operator `>>` wczytujący tę liczbę ze strumienia typu `istream`. Umawiamy się, że liczbę wymierną zapisujemy w postaci `p/q`, np. `3/7`.

Konstruktor i operator `>>` powinny działać poprawnie również jeśli podane p i q nie są względnie pierwsze lub q jest ujemne.

Napisz program `rat`, który wczytuje dwie liczby wymierne, a następnie wypisuje w kolejnych liniach:

1. podane liczby w postaci dziesiętnej
2. liczby przeciwne do podanych
3. podane liczby w kolejności niemalejącej
4. sumę oraz iloczyn podanych liczb

Oto przykładowy wynik działania programu:

```
./rat
2/4 -1/3
0.5 -0.333333
-1/3 1/2
-1/2 1/3
1/6 -1/6
```

```
#include <iostream>

using namespace std;

class Rat {
```

```
  int p,q;
public:
  Rat (int p=0,int q=1);
  operator double () const;
  Rat operator - () const;
  Rat &operator ++ ();
  Rat &operator += (const Rat &rat);
  Rat &operator *= (const Rat &rat);
  int numerator () const;
  int denominator () const;
  friend bool operator < (const Rat &first,const Rat &second);
  friend Rat operator + (const Rat &first,const Rat &second);
  friend Rat operator * (const Rat &first,const Rat &second);
  friend ostream &operator << (ostream &stream,const Rat &rat);
  friend istream &operator >> (istream &stream,Rat &rat); };

Rat::Rat (int p,int q): p (p),q (q) {
  while (p) {
    int t=q%p;
    q=p;
    p=t; }
  q=(this->q>0 ? 1 : -1)*abs (q);
  this->p/=q;
  this->q/=q; }

Rat::operator double () const {
  return (double) p/q; }

Rat Rat::operator - () const {
  return Rat (-p,q); }

Rat &Rat::operator ++ () {
  p+=q;
  return *this; }

Rat &Rat::operator += (const Rat &rat) {
  return *this=*this+rat; }

Rat &Rat::operator *= (const Rat &rat) {
  return *this=*this*rat; }

int Rat::numerator () const {
  return p; }

int Rat::denominator () const {
  return q; }

bool operator < (const Rat &first,const Rat &second) {
  return first.p*second.q<first.q*second.p; }
```

```
Rat operator + (const Rat &first,const Rat &second) {
  return Rat (first.p*second.q+first.q*second.p,first.q*second.q); }

Rat operator * (const Rat &first,const Rat &second) {
  return Rat (first.p*second.p,first.q*second.q); }

ostream &operator << (ostream &stream,const Rat &rat) {
  return stream<<rat.p<<'/'<<rat.q; }

istream &operator >> (istream &stream,Rat &rat) {
  char c;
  stream>>rat.p>>c>>rat.q;
  rat=Rat (rat.p,rat.q);
  return stream; }

int main () {
  Rat a,b;
  cin>>a>>b;
  cout<<(double) a<<" "<<(double) b<<endl;
  cout<<(a<b ? a : b)<<" "<<(a<b ? b : a)<<endl;
  cout<<-a<<" "<<-b<<endl;
  cout<<a+b<<" "<<a*b<<endl;
  return 0; }
```

4.1.10 Sentence

Napisz klasę `Sentence` reprezentującą zdanie, czyli ciąg słów zakończony kropką. Przyjmij, że znaki interpunkcyjne umieszczane są bezpośrednio za ostatnią literą wyrazu, np. `ten, który`. Zaimplementuj:

- Funkcję `words` zwracającą liczbę słów w zdaniu.
- Operator `<<` wypisujący zdanie do strumienia typu `ostream`. W wypisywanym zdaniu wyrazy powinny być oddzielone pojedynczymi spacjami.
- Operator `>>` wczytujący zdanie ze strumienia typu `istream`. We wczytywanym zdaniu wyrazy mogą być oddzielone dowolną liczbą dowolnych znaków białych.

Pewien plik zawiera jedynie zdania, z wyrazami oddzielonymi różnymi znakami białymi. Napisz program `sentence`, który wczyta wszystkie zdania z tego pliku i wypisze je w kolejnych liniach, po jedno zdanie w linii, oddzielając wyrazy pojedynczymi spacjami. Na końcu każdej linii powinna być podana liczba wyrazów w zdaniu.

```
#include <iostream>
#include <string>

using namespace std;

class Sentence {
public:
  int words ();
```

```
  friend ostream &operator<< (ostream &stream,const Sentence &sentence);
  friend istream &operator>> (istream &stream,Sentence &sentence);
private:
  string sentence;
  int number; };

int Sentence::words () {
  return number; }

ostream &operator<< (ostream &stream,const Sentence &sentence) {
  return stream<<sentence.sentence; }

istream &operator>> (istream &stream,Sentence &sentence) {
  sentence.sentence.clear ();
  sentence.number=0;
  string word;
  do {
    if (!(stream>>word))
      break;
    sentence.sentence+=word+" ";
    ++sentence.number; }
  while (word [word.size ()-1]!='.');
  if (sentence.sentence.size ())
    sentence.sentence.erase (sentence.sentence.size ()-1,1);
  return stream; }

int main () {
  Sentence sentence;
  while (cin>>sentence)
    cout<<sentence<<" "<<sentence.words ()<<endl;
  return 0; }
```

4.1.11 Vector

Napisz klasę `Vector` reprezentującą dwuwymiarowy wektor. Zaimplementuj:

- Konstruktor bezargumentowy inicjalizujący współrzędne kartezjańskie wektora zerami oraz konstruktor dwuargumentowy inicjalizujący je swoimi argumentami.
- Funkcję składową `length` zwracającą długość wektora.
- Jednoargumentowy operator `-`.
- Operatory dodawania i odejmowania dwóch wektorów oraz operatory `+=` i `-=`.
- Operatory lewo- i prawostronnego mnożenia wektora przez liczbę oraz operator `*=`.
- Operator `*` iloczynu skalarnego dwóch wektorów.
- Operator `<<` wypisywania współrzędnych wektora do strumienia typu `ostream` oraz operator `>>` wczytywania współrzędnych ze strumienia typu `istream`. Współrzędne wektora zapisujemy jako dwie liczby oddzielone spacją.

```
#include <cmath>
#include <iostream>

using namespace std;

class Vector {
public:
  Vector ();
  Vector (double x,double y);
  double length () const;
  Vector operator - () const;
  Vector &operator += (const Vector &vector);
  Vector &operator -= (const Vector &vector);
  Vector &operator *= (double number);
  friend Vector operator * (double number,const Vector &vector);
  friend Vector operator * (const Vector &vector,double number);
  friend double operator * (const Vector &first,const Vector &second);
  friend Vector operator + (const Vector &first,const Vector &second);
  friend Vector operator - (const Vector &first,const Vector &second);
  friend ostream &operator << (ostream &stream,const Vector &vector);
  friend istream &operator >> (istream &stream,Vector &vector);
private:
  double x,y; };

Vector::Vector (): x (),y () {}

Vector::Vector (double x,double y): x (x),y (y) {}

double Vector::length () const {
  return sqrt (x*x+y*y); }

Vector Vector::operator - () const {
  return Vector (-x,-y); }

Vector &Vector::operator += (const Vector &vector) {
  x+=vector.x;
  y+=vector.y;
  return *this; }

Vector &Vector::operator -= (const Vector &vector) {
  x-=vector.x;
  y-=vector.y;
  return *this; }

Vector &Vector::operator *= (double number) {
  x*=number;
  y*=number;
  return *this; }
```

```
Vector operator * (double number,const Vector &vector) {
  return Vector (number*vector.x,number*vector.y); }

Vector operator * (const Vector &vector,double number) {
  return Vector (vector.x*number,vector.y*number); }

double operator * (const Vector &first,const Vector &second) {
  return first.x*second.x+first.y*second.y; }

Vector operator + (const Vector &first,const Vector &second) {
  return Vector (first.x+second.x,first.y+second.y); }

Vector operator - (const Vector &first,const Vector &second) {
  return Vector (first.x-second.x,first.y-second.y); }

ostream &operator << (ostream &stream,const Vector &vector) {
  return stream<<vector.x<<' '<<vector.y; }

istream &operator >> (istream &stream,Vector &vector) {
  return stream>>vector.x>>vector.y; }
```

4.1.12 Class - przykład

Poniższy przykład ilustruje wywołania konstruktorów, destruktorów, operatorów i innych funkcji. Posłużono się tu metodą gadających funkcji, to znaczy każda funkcja wypisuje komunikat, że została wywołana.

Zdefiniowana jest następująca klasa oraz dwie funkcje:

```
#include <iostream>

using namespace std;

class Class {
public:
  Class (int i=0);
  Class (const Class &c);
  Class &operator= (const Class &c);
 ~Class ();
  int i; };

Class::Class (int i): i (i) {
  cout<<"To ja, konstruktor domyslny: i="<<i<<endl; }

Class::Class (const Class &c): i (c.i) {
  cout<<"To ja, konstruktor kopiujacy: i="<<i<<endl; }

Class &Class::operator = (const Class &c) {
  i=c.i;
  cout<<"To ja, operator przypisania: i="<<i<<endl;
  return *this; }
```

```
Class::~Class () {
  cout<<"To ja, destruktor: i="<<i<<endl; }

Class value (Class c) {
  cout<<"To ja, funkcja przekazujaca przez wartosc: i="<<c.i<<endl;
  return c; }

const Class &reference (const Class &c) {
  cout<<"To ja, funkcja przekazujaca przez referencje: i="<<c.i<<endl;
  return c; }
```

1. Wywołanie programu

   ```
   int main () {
     Class a,b (3),c=7,d (b),e=c;
     return 0; }
   ```

   daje wynik:

   ```
   To ja, konstruktor domyslny: i=0
   To ja, konstruktor domyslny: i=3
   To ja, konstruktor domyslny: i=7
   To ja, konstruktor kopiujacy: i=3
   To ja, konstruktor kopiujacy: i=7
   To ja, destruktor: i=7
   To ja, destruktor: i=3
   To ja, destruktor: i=7
   To ja, destruktor: i=3
   To ja, destruktor: i=0
   ```

2. Wywołanie programu

   ```
   int main () {
     Class a;
     a=7;
     return 0; }
   ```

   daje wynik:

   ```
   To ja, konstruktor domyslny: i=0
   To ja, konstruktor domyslny: i=7
   To ja, operator przypisania: i=7
   To ja, destruktor: i=7
   To ja, destruktor: i=7
   ```

3. Wywołanie programu

   ```
   int main () {
     Class a;
     value (a);
     return 0; }
   ```

daje wynik:

```
To ja, konstruktor domyslny: i=0
To ja, konstruktor kopiujacy: i=0
To ja, funkcja przekazująca przez wartosc: i=0
To ja, konstruktor kopiujacy: i=0
To ja, destruktor: i=0
To ja, destruktor: i=0
To ja, destruktor: i=0
```

4. Wywołanie programu

```
int main () {
  value (3);
  return 0; }
```

daje wynik:

```
To ja, konstruktor domyslny: i=3
To ja, funkcja przekazujaca przez wartosc: i=3
To ja, konstruktor kopiujacy: i=3
To ja, destruktor: i=3
To ja, destruktor: i=3
```

5. Wywołanie programu

```
int main () {
  Class a;
  reference (a);
  return 0; }
```

daje wynik:

```
To ja, konstruktor domyslny: i=0
To ja, funkcja przekazujaca przez referencje: i=0
To ja, destruktor: i=0
```

6. Wywołanie programu

```
int main () {
  reference (7);
  return 0; }
```

daje wynik:

```
To ja, konstruktor domyslny: i=7
To ja, funkcja przekazujaca przez referencje: i=7
To ja, destruktor: i=7
```

4.1.13 Resistor

Napisz klasę `Resistor` reprezentującą opornik o zadanym oporze. Zaimplementuj:

- Funkcję składową `set` ustawiającą opór opornika. Deklaracja:

  ```
  void Resistor::set (double resistance);
  ```

- Funkcję zaprzyjaźnioną `resistance` zwracającą opór opornika. Deklaracja:

  ```
  double resistance (const Resistor &resistor);
  ```

- Funkcje zaprzyjaźnione `serial` i `parallel` tworzące oporniki odpowiadające szeregowemu i równoległemu połączeniu dwóch oporników. Deklaracje:

  ```
  Resistor serial (const Resistor &first,const Resistor &second);
  Resistor parallel (const Resistor &first,const Resistor &second);
  ```

Napisz program `resistor`, który wczytuje ze standardowego wejścia dwa opory, a następnie wypisuje na standardowe wyjście opór powstały z ich szeregowego oraz równoległego połączenia.

```
#include <iostream>

using namespace std;

class Resistor {
  double r;
public:
  void set (double r);
  friend double resistance (const Resistor &resistor);
  friend Resistor serial (const Resistor &first,const Resistor &second);
  friend Resistor parallel (const Resistor &first,const Resistor &second); };

void Resistor::set (double r) {
  Resistor::r=r; }

double resistance (const Resistor &resistor) {
  return resistor.r; }

Resistor serial (const Resistor &first,const Resistor &second) {
  Resistor resistor;
  resistor.r=first.r+second.r;
  return resistor; }

Resistor parallel (const Resistor &first,const Resistor &second) {
  Resistor resistor;
  resistor.r=1/(1/first.r+1/second.r);
  return resistor; }

int main () {
  double r1,r2;
```

```
  cin>>r1>>r2;
  Resistor R1,R2;
  R1.set (r1);
  R2.set (r2);
  cout<<resistance (serial (R1,R2))<<" "<<resistance (parallel (R1,R2))<<endl;
  return 0; }
```

4.2 Dziedziczenie

4.2.1 Figure1 - przykład

W poniższym programie zdefiniowana jest prosta klasa reprezentująca figurę o zadanym kolorze.

```
#include <iostream>

using namespace std;

class Figure {
  const string color;
public:
  Figure (const string &color);
  const string &getColor () const; };

Figure::Figure (const string &color): color (color) {}

const string &Figure::getColor () const {
  return color; }

int main () {
  Figure figure ("red");
  cout<<figure.getColor ()<<endl;
  return 0; }
```

Wynikiem działania tego programu jest

```
red
```

4.2.2 Figure2 - przykład

Poniższy program ilustruje dziedziczenie danych i funkcji składowych, wywołanie konstruktorów klas macierzystych oraz użycie referencji i wskaźników do klas macierzystych.

```
#include <iostream>

using namespace std;

class Figure {
  const string color;
public:
  Figure (const string &color);
  const string &getColor () const; };
```

```
Figure::Figure (const string &color): color (color) {}

const string &Figure::getColor () const {
  return color; }

class Circle: public Figure {
  const double r;
public:
  Circle (const string &color,double r);
  double getRadius () const; };

Circle::Circle (const string &color,double r): Figure (color),r (r) {}

double Circle::getRadius () const {
  return r; }

class Rectangle: public Figure {
  const double a,b;
public:
  Rectangle (const string &color,double a,double b);
  double getFirstEdge () const;
  double getSecondEdge () const; };

Rectangle::Rectangle (const string &color,double a,double b):
   Figure (color),a (a),b (b) {}

double Rectangle::getFirstEdge () const {
  return a; }

double Rectangle::getSecondEdge () const {
  return b; }

int main () {
  Circle circle ("red",1);
  cout<<"circle "<<circle.getColor ()<<" "<<circle.getRadius ()<<endl;
  Rectangle rectangle ("green",3,4);
  cout<<"rectangle "<<rectangle.getColor ()<<" "
      <<rectangle.getFirstEdge ()<<" "<<rectangle.getSecondEdge ()<<endl;
  Figure &reference=circle;
  cout<<"circle "<<reference.getColor ()<<endl;
  Figure *pointer=&circle;
  cout<<"circle "<<pointer->getColor ()<<endl;
  return 0; }
```

Wynikiem jego działania jest

```
circle red 1
rectangle green 3 4
circle red
circle red
```

4.2.3 Figure3 - przykład

Kolejny program wykorzystuje klasy abstrakcyjne i funkcje wirtualne.

```
#include <iostream>
#include <cmath>

using namespace std;

class Figure {
  const string color;
public:
  Figure (const string &color);
  const string &getColor () const;
  virtual double getArea () const=0; };

Figure::Figure (const string &color): color (color) {}

const string &Figure::getColor () const {
  return color; }

class Circle: public Figure {
  const double r;
public:
  Circle (const string &color,double r);
  double getArea () const;
  double getRadius () const; };

Circle::Circle (const string &color,double r): Figure (color),r (r) {}

double Circle::getArea () const {
  return M_PI*r*r; }

double Circle::getRadius () const {
  return r; }

class Rectangle: public Figure {
  const double a,b;
public:
  Rectangle (const string &color,double a,double b);
  double getArea () const;
  double getFirstEdge () const;
  double getSecondEdge () const; };

Rectangle::Rectangle (const string &color,double a,double b):
   Figure (color),a (a),b (b) {}

double Rectangle::getArea () const {
  return a*b; }
```

```
double Rectangle::getFirstEdge () const {
  return a; }

double Rectangle::getSecondEdge () const {
  return b; }

int main () {
  Circle circle ("red",1);
  cout<<"circle "<<circle.getColor ()<<" "
      <<circle.getRadius ()<<" "<<circle.getArea ()<<endl;
  Rectangle rectangle ("green",3,4);
  cout<<"rectangle "<<rectangle.getColor ()<<" "
      <<rectangle.getFirstEdge ()<<" "<<rectangle.getSecondEdge ()<<" "
      <<rectangle.getArea ()<<endl;
  Figure &reference=circle;
  cout<<"circle "<<reference.getColor ()<<" "<<reference.getArea ()<<endl;
  Figure *pointer=&circle;
  cout<<"circle "<<pointer->getColor ()<<" "<<pointer->getArea ()<<endl;
  return 0; }
```

Wynikiem jego działania jest

```
circle red 1 3.14159
rectangle green 3 4 12
circle red 3.14159
circle red 3.14159
```

4.2.4 Figure4 - przykład

W następnym przykładzie zaimplementowano operator pisania do strumienia.

```
#include <iostream>
#include <cmath>

using namespace std;

class Figure {
  const string color;
  virtual ostream &print (ostream &stream) const=0;
public:
  Figure (const string &color);
  const string &getColor () const;
  virtual double getArea () const=0;
  friend ostream &operator << (ostream &stream,const Figure &figure); };

Figure::Figure (const string &color): color (color) {}

const string &Figure::getColor () const {
  return color; }
```

```
ostream &operator << (ostream &stream,const Figure &figure) {
  return figure.print (stream); }

class Circle: public Figure {
  const double r;
  ostream &print (ostream &stream) const;
public:
  Circle (const string &color,double r);
  double getArea () const;
  double getRadius () const; };

ostream &Circle::print (ostream &stream) const {
  return stream<<"circle "<<getColor ()<<" "<<r; }

Circle::Circle (const string &color,double r): Figure (color),r (r) {}

double Circle::getArea () const {
  return M_PI*r*r; }

double Circle::getRadius () const {
  return r; }

class Rectangle: public Figure {
  const double a,b;
  ostream &print (ostream &stream) const;
public:
  Rectangle (const string &color,double a,double b);
  double getArea () const;
  double getFirstEdge () const;
  double getSecondEdge () const; };

ostream &Rectangle::print (ostream &stream) const {
  return stream<<"rectangle "<<getColor ()<<" "<<a<<" "<<b; }

Rectangle::Rectangle (const string &color,double a,double b):
   Figure (color),a (a),b (b) {}

double Rectangle::getArea () const {
  return a*b; }

double Rectangle::getFirstEdge () const {
  return a; }

double Rectangle::getSecondEdge () const {
  return b; }

int main () {
  Circle circle ("red",1);
  cout<<circle<<" "<<circle.getArea ()<<endl;
```

```
  Rectangle rectangle ("green",3,4);
  cout<<rectangle<<" "<<rectangle.getArea ()<<endl;
  Figure &reference=circle;
  cout<<reference<<" "<<reference.getArea ()<<endl;
  Figure *pointer=&circle;
  cout<<*pointer<<" "<<pointer->getArea ()<<endl;
  return 0; }
```

Wynikiem działania tego programu jest

```
circle red 1 3.14159
rectangle green 3 4 12
circle red 1 3.14159
circle red 1 3.14159
```

4.2.5 Function

Zadeklaruj abstrakcyjną klasę `Function` reprezentującą funkcję rzeczywistą zmiennej rzeczywistej. Zadeklaruj w niej wirtualny operator wywołania funkcji, który umożliwia obliczenie wartości dowolnej pochodnej funkcji w dowolnym punkcie. Jeżeli rząd pochodnej nie jest podany, to operator powinien zwracać wartość samej funkcji.

Z klasy `Function` wyprowadź klasy `Exponential` oraz `Sinus` reprezentujące odpowiednio funkcje $\exp(\alpha x)$ oraz $\sin(\omega x)$. Zaimplmentuj w nich konstruktory inicjalizujące wartości stałych α i ω oraz konkretyzacje operatora wywołania funkcji odziedziczonego z klasy `Function`.

Wymienione klasy skonstruuj w taki sposób, aby program:

```
int main () {
  Exponential exponential (2.);
  Sine sine (3.);
  Function &expo=exponential;
  Function &sinus=sine;
  cout<<expo (1.)<<endl;
  cout<<sinus (2.,1)<<endl;
  return 0; }
```

wypisywał odpowiednio wartość funkcji $\exp(2x)$ w punkcie 1. oraz pierwszą pochodną funkcji $\sin(3x)$ w punkcie 2.

```
#include <iostream>
#include <cmath>

using namespace std;

class Function {
public:
  virtual double operator () (double x,int n=0) const=0; };

class Exponential: public Function {
  const double alpha;
public:
  Exponential (double alpha);
```

```
  virtual double operator () (double x,int n=0) const; };

Exponential::Exponential (double alpha): alpha (alpha) {}

double Exponential::operator () (double x,int n) const {
  return pow (alpha,n)*exp (alpha*x); }

class Sine: public Function {
  const double omega;
public:
  Sine (double omega);
  virtual double operator () (double x,int n=0) const; };

Sine::Sine (double omega): omega (omega) {}

double Sine::operator () (double x,int n) const {
  return pow (omega,n)*(n%2 ? (2-n%4)*cos (omega*x) : (1-n%4)*sin (omega*x)); }

int main () {
  Exponential exponential (2.);
  Sine sine (3.);
  Function &expo=exponential;
  Function &sinus=sine;
  cout<<expo (1.)<<endl;
  cout<<sinus (2.,1)<<endl;
  return 0; }
```

```
7.38906
2.88051
```

4.2.6 Worker

Napisz klasę `Worker` pamiętającą nazwisko i pensję pracownika. Zaimplementuj:

- Konstruktor, który umożliwia utworzenie obiektu reprezentującego pracownika o zadanym nazwisku i pensji. W przypadku niepodania pensji powinno być przyjmowane minimalne wynagrodzenie w roku 2011, czyli 1386zł.
- Funckję `increase` umożliwiającą przyznanie pracownikowi podwyżki o zadaną kwotę.
- Operator `<<` wypisujący do strumienia typu `ostream` nazwisko i pensję.

Z klasy tej wyprowadź klasę `Chief` reprezentującą szefa. Szef nie ma domyślnego wynagrodzenia, posiada natomiast samochód służbowy. Oprócz nazwiska i pensji operator `<<` powinien wypisywać markę samochodu.

Klasy `Worker` i `Chief` powinny być napisane tak, aby wykonanie następującego programu:

```
int main () {
  Worker *worker=new Worker ("Smith");
  Worker *chief=new Chief ("Brown",20000.,"Lamborghini Diablo");
  cout<<*worker<<endl<<*chief<<endl;
  chief->increase (5000.);
```

```
  cout<<*chief<<endl;
  delete chief;
  delete worker;
  return 0; }
```

dawało taki wydruk:

```
Smith earns 1386 PLN and works hard
Brown earns 20000 PLN and drives Lamborghini Diablo
Brown earns 25000 PLN and drives Lamborghini Diablo
```

4.2.7 Sequence

Zadeklaruj abstrakcyjną klasę `Sequence` reprezentującą ciąg matematyczny. Zadeklaruj w niej publiczny operator indeksowania zwracający żądany wyraz ciągu.

Z klasy tej wyprowadź klasę `Arithmetic` reprezentującą ciąg arytmetyczny

$$a_n = a_0 + n\Delta$$

Zdefiniuj dla niej konstruktor umożliwiający nadanie wartości parametrom a_0 oraz Δ.

Z klasy `Sequence` wywiedź ponadto klasę `Fibonacci` reprezentującą ciąg Fibonacciego, określony zależnością rekurencyjną

$$a_0 = 0 \qquad a_1 = 1 \qquad a_n = a_{n-1} + a_{n-2}$$

Klasy te powinny być napisane tak, aby można je było wykorzystać w programie

```
int main () {
  Sequence *arithmetic=new Arithmetic (2.,3.);
  Sequence *fibonacci=new Fibonacci;
  for (int n=0;n<10;n++)
    cout<<(*arithmetic) [n]<<" ";
  cout<<endl;
  for (int n=0;n<10;n++)
    cout<<(*fibonacci) [n]<<" ";
  cout<<endl;
  delete fibonacci;
  delete arithmetic;
  return 0; }
```

Jego wykonanie powinno dawać następujący wydruk:

```
2 5 8 11 14 17 20 23 26 29
0 1 1 2 3 5 8 13 21 34
```

```
#include <iostream>
#include <cmath>

using namespace std;

class Sequence {
public:
```

```
  virtual double operator [] (int n) const=0; };

class Arithmetic:public Sequence {
  const double first,delta;
public:
  Arithmetic (double first,double delta);
  virtual double operator [] (int n) const; };

Arithmetic::Arithmetic (double first,double delta): first (first),delta (delta) {}

double Arithmetic::operator [] (int n) const {
  return first+n*delta; }

class Fibonacci:public Sequence {
public:
  virtual double operator [] (int n) const; };

double Fibonacci::operator [] (int n) const {
  if (n<2)
    return n;
  double a=0,b=1,c;
  for (int k=2;k<n;k++) {
    c=b+a;
    a=b;
    b=c; }
  return b+a; }

int main () {
  Sequence *arithmetic=new Arithmetic (2.,3.);
  Sequence *fibonacci=new Fibonacci;
  for (int n=0;n<10;n++)
    cout<<(*arithmetic) [n]<<" ";
  cout<<endl;
  for (int n=0;n<10;n++)
    cout<<(*fibonacci) [n]<<" ";
  cout<<endl;
  delete fibonacci;
  delete arithmetic;
  return 0; }
```

4.2.8 Scale

Przy rysowaniu wykresu na ekranie monitora pojawia się konieczność przeliczania abstrakcyjnych współrzędnych wykresu na współrzędne punktów na ekranie. Można to robić tak, aby powstał wykres w skali liniowej, logarytmicznej lub innej. Transformacja współrzędnych jest dana osobno dla osi odciętych i rzędnych. Jest ona jednoznacznie wyznaczona przez przedział współrzędnych abstrakcyjnych, w którym chcemy sporządzić wykres, oraz odpowiadający mu przedział współrzędnych ekranowych. Przykładowo, jeżeli wykres przedstawiający abstrakcyjne współrzędne w przedziale od 1 do 10 ma rozciągać się na ekranie od punktu 2 do punktu 20, to współrzędne ekranowe x_s wyrażają się przez współrzędne abstrakcyjne x_a wzorem $x_s = 2x_a$.

Napisz abstrakcyjną klasę `Scale` reprezentującą transformację pomiędzy współrzędnymi abstrakcyjnymi a ekranowymi dla pojedynczej osi. Zaimplementuj:

- Wirtualną abstrakcyjną funkcję składową `x` zwracającą współrzędną abstrakcyjną odpowiadającą współrzędnej ekranowej zadanej argumentem funkcji.
- Wirtualną abstrakcyjną funkcję składową `X` zwracającą współrzędną ekranową odpowiadającą współrzędnej abstrakcyjnej zadanej argumentem funkcji.

Z klasy `Scale` wywiedź klasy `Linear` oraz `Logarithmic` reprezentujące odpowiednio skalę liniową i logarytmiczną. Zaimplementuj w nich konkretyzacje funkcji `x` i `X` oraz czteroargumentowe konstruktory umożliwiające zadanie zakresów współrzędnych abstrakcyjnych i ekranowych.

```
class Scale {
public:
  virtual double x (double X) const=0;
  virtual double X (double x) const=0; };

class Linear: public Scale {
  double a,b,A,B;
public:
  Linear (double minx,double maxx,double minX,double maxX);
  double x (double X) const;
  double X (double x) const; };

Linear::Linear (double minx,double maxx,double minX,double maxX) {
  a=(maxx-minx)/(maxX-minX);
  b=minx-a*minX;
  A=(maxX-minX)/(maxx-minx);
  B=minX-A*minx; }

double Linear::x (double X) const {
  return b+a*X; }

double Linear::X (double x) const {
  return B+A*x; }

class Logarithmic: public Scale {
  double a,b,A,B;
public:
  Logarithmic (double minx,double maxx,double minX,double maxX);
  double x (double X) const;
  double X (double x) const; };

Logarithmic::Logarithmic (double minx,double maxx,double minX,double maxX) {
  a=(log (maxx)-log (minx))/(maxX-minX);
  b=minx/exp (a*minX);
  A=(maxX-minX)/(log (maxx)-log (minx));
  B=minX-A*log (minx); }

double Logarithmic::x (double X) const {
```

```
  return b*exp (a*X); }

double Logarithmic::X (double x) const {
  return B+A*log (x); }
```

4.2.9 Optics

W optyce macierzowej każdy układ elementów optycznych ustawionych wzdłuż jednej osi opisany jest pewną macierzą 2×2. W szczególności pusty odcinek o długości d oraz cienka soczewka o ogniskowej f reprezentowane są odpowiednio macierzami

$$\hat{S} = \begin{bmatrix} 1 & d \\ 0 & 1 \end{bmatrix} \qquad \hat{L} = \begin{bmatrix} 1 & 0 \\ -1/f & 1 \end{bmatrix}$$

Jeżeli na osi optycznej bezpośrednio za elementem opisanym macierzą $\hat{M}_1$ stoi element opisany macierzą $\hat{M}_2$, to układowi temu odpowiada macierz $\hat{M} = \hat{M}_2\hat{M}_1$. Promień światła opisany jest dwuwymiarowym wektorem

$$\vec{u} = \begin{bmatrix} y \\ \alpha \end{bmatrix}$$

gdzie y oraz α są odpowiednio odległością promienia od osi optycznej oraz kątem między promieniem a osią. Promień opisany wektorem $\vec{u}$ po przejściu przez układ opisany macierzą $\hat{M}$ przechodzi w promień opisany wektorem $\vec{u}' = \hat{M}\vec{u}$.

Napisz klasę `Ray` reprezentującą promień światła. Zaimplementuj w niej:

- Bezargumentowy konstruktor zerujący y oraz α
- Dwuargumentowy konstruktor inicjalizujący y oraz α swoimi argumentami
- Funkcje składowe `getDistance` oraz `getAngle` zwracające odpowiednio y i α

Napisz klasę `System` opisującą układ elementów optycznych. Zaimplementuj w niej bezargumentowy konstruktor inicjalizujący macierz układu macierzą jednostkową.

Jako funkcję zaprzyjaźnioną z klasą `System` zaimplementuj operator `*` odpowiadający składaniu dwóch układów ustawionych bezpośrednio jeden za drugim. Jako funkcję zaprzyjaźnioną z obiema klasami zaimplementuj operator `*` odpowiadający działaniu układu na promień.

Z klasy `System` wywiedź klasę `Empty` reprezentującą pusty odcinek osi optycznej. Zaimplementuj w niej jednoargumentowy konstruktor inicjalizujący długość odcinka swoim argumentem. Z klasy `System` wywiedź też klasę `Lens` reprezentującą soczewkę. Zaimplementuj w niej jednoargumentowy konstruktor inicjalizujący ogniskową swoim argumentem.

Wszystkie klasy i funkcje powinny być napisane tak, aby działał następujący program:

```
int main () {
  Ray ray=Lens (5)*Empty (7)*Lens (9)*Ray (0.1,0);
  cout<<ray.getAngle ()<<" "<<ray.getDistance ()<<endl;
  return 0; }
```

Program ten znajduje parametry y i α promienia o początkowym $y = 0.1$ i $\alpha = 0$ po przejściu kolejno przez soczewkę o ogniskowej 9, pusty odcinek o długości 7 i soczewkę o ogniskowej 5.

```
#include <iostream>
#include <cmath>
```

```
using namespace std;

class Ray;
class System;

class Ray {
  double u [2];
public:
  Ray ();
  Ray (double y,double alpha);
  double getDistance () const;
  double getAngle () const;
  friend Ray operator * (const System &s,const Ray &r); };

Ray::Ray () {
  u [0]=0;
  u [1]=0; }

Ray::Ray (double y,double alpha) {
  u [0]=y;
  u [1]=alpha; }

double Ray::getDistance () const {
  return u [0]; }

double Ray::getAngle () const {
  return u [1]; }

class System {
protected:
  double m [2][2];
public:
  System ();
  friend System operator * (const System &s1,const System &s2);
  friend Ray operator * (const System &s,const Ray &r); };

System::System () {
  for (int i=0;i<2;i++)
    for (int j=0;j<2;j++)
      m [i][j]=(i==j ? 1 :0); }

System operator * (const System &s1,const System &s2) {
  System s;
  for (int i=0;i<2;i++)
    for (int j=0;j<2;j++) {
      s.m [i][j]=0;
      for (int k=0;k<2;k++)
        s.m [i][j]+=s1.m [i][k]*s2.m [k][j]; }
  return s; }
```

```
Ray operator * (const System &s,const Ray &r) {
  Ray ray;
  for (int i=0;i<2;i++) {
    ray.u [i]=0;
    for (int j=0;j<2;j++)
      ray.u [i]+=s.m [i][j]*r.u [j]; }
  return ray; }

class Empty: public System {
public:
  Empty (double d); };

Empty::Empty (double d) {
  m [0][0]=1;
  m [0][1]=d;
  m [1][0]=0;
  m [1][1]=1; }

class Lens: public System {
public:
  Lens (double f); };

Lens::Lens (double f) {
  m [0][0]=1;
  m [0][1]=0;
  m [1][0]=-1/f;
  m [1][1]=1; }

int main () {
  Ray ray=Lens (5)*Empty (7)*Lens (9)*Ray (0.1,0);
  cout<<ray.getDistance ()<<" "<<ray.getAngle ()<<endl;
  return 0; }
```

5 Pojemniki STL

5.1 Vector

5.1.1 Vector

Poniższy program wczytuje nieznaną liczbę liczb całkowitych do wektora, a potem wypisuje je w kolejności rosnącej raz przy pomocy indeksów, a raz przy pomocy iteratorów.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main () {
  vector <int> items;
```

```
  int item;
  while (cin>>item)
    items.push_back (item);
  sort (items.begin (),items.end ());
  for (int index=0;index<items.size ();++index)
    cout<<items [index]<<' ';
  cout<<endl;
  for (vector <int>::iterator iterator=items.begin ();
       iterator!=items.end ();
       ++iterator)
    cout<<*iterator<<' ';
  cout<<endl;
  return 0; }
```

5.1.2 Spreadsheet

Pewien plik tekstowy ma następującą strukturę:

```
-1.0e+00  2.0e+01
 3.0e+01 -4.0e+00
```

Każda liczba zapisana jest w formacie naukowym na 8 znakach, a pomiędzy liczbami znajdują się pojedyncze spacje. Plik nie zawiera pustych linii. Liczba kolumn ani wierszy nie jest z góry znana. Przyjmujemy jednak, że w pliku znajduje się przynajmniej jedna liczba.

Napisz program `spreadsheet`, który na końcu każdego wiersza dopisze sumę wszystkich liczb z tego wiersza, a na końcu każdej kolumny dopisze sumę wszystkich liczb z tej kolumny. Oprócz tego na przecięciu nowodopisanego wiersza i nowodopisanej kolumny powinna znaleźć się suma wszystkich liczb z oryginalnego pliku. W podanym przykładzie wynik działania programu powinien wyglądać następująco:

```
-1.0e+00  2.0e+01  1.9e+01
 3.0e+01 -4.0e+00  2.6e+01
 2.9e+01  1.6e+00  4.5e+01
```

Program powinien czytać dane ze standardowego wejścia, a wynik wypisywać na standardowe wyjście.

```
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>

using namespace std;

int main () {
  cout<<scientific<<setprecision (1);
  vector <double> sums;
  { string line;
    getline (cin,line);
    istringstream stream (line);
```

```
    double value,sum=0.;
    while (stream>>value) {
      cout<<setw (8)<<value<<" ";
      sums.push_back (value);
      sum+=value; }
    cout<<setw (8)<<sum<<endl; }
  { string line;
    while (getline (cin,line)) {
      istringstream stream (line);
      double value,sum=0.;
      for (int i=0;i!=sums.size ();++i) {
        stream>>value;
        cout<<setw (8)<<value<<" ";
        sums [i]+=value;
        sum+=value; }
      cout<<setw (8)<<sum<<endl; }}
  { double sum=0.;
    for (int i=0;i!=sums.size ();++i) {
      cout<<setw (8)<<sums [i]<<" ";
      sum+=sums [i]; }
    cout<<setw (8)<<sum<<endl; }
  return 0; }
```

5.1.3 Names

Korzystając z klasy `Name` napisz program `names`, który wczytuje ze standardowego wejścia listę imion i nazwisk, a potem wypisuje je na standardowe wyjście najpierw w kolejności alfabetycznej, a następnie w kolejności alfabetycznej inicjałów.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>

using namespace std;

class Name {
private:
  string given,family;
public:
  Name ();
  Name (const string &given,const string &family);
  string initials () const;
  friend bool operator< (const Name &first,const Name &second);
  friend ostream &operator<< (ostream &stream,const Name &Name);
  friend istream &operator>> (istream &stream,Name &Name); };

Name::Name (): given (),family () {}

Name::Name (const string &given,const string &family):
```

```
  given (given),family (family) {}

string Name::initials () const {
  return given.substr (0,1)+family.substr (0,1); }

bool operator< (const Name &first,const Name &second) {
  return first.family<second.family ||
        (first.family==second.family && first.given<second.given); }

ostream &operator<< (ostream &stream,const Name &Name) {
  return stream<<Name.given<<' '<<Name.family; }

istream &operator>> (istream &stream,Name &Name) {
  return stream>>Name.given>>Name.family; }

bool compare (const Name &first,const Name &second) {
  return first.initials ()<second.initials (); }

int main () {
  vector <Name> names;
  Name name;
  while (cin>>name)
    names.push_back (name);
  sort (names.begin (),names.end ());
  for (vector <Name>::iterator iterator=names.begin ();
       iterator!=names.end ();
       ++iterator)
    cout<<*iterator<<endl;
  cout<<endl;
  sort (names.begin (),names.end (),compare);
  for (vector <Name>::iterator iterator=names.begin ();
       iterator!=names.end ();
       ++iterator)
    cout<<*iterator<<endl;
  return 0; }
```

5.1.4 Lagrange

Przez n punktów (x_i, y_i), gdzie $i = 0, ..., n-1$, przechodzi dokładnie jeden wielomian stopnia $n-1$, zwany wielomianem interpolacyjnym Lagrange'a. Dany jest on wzorem

$$L(x) = \sum_{i=0}^{n-1} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

Napisz procedurę `lagrange` znajdującą wartość tego wielomianu w zadanym punkcie x. Deklaracja:

```
double lagrange (const vector <pair <double,double> > &points,double x);
```

Argument	Wejście	Wyjście
`lagrange`		$L(x)$
`points`	(x_i, y_i)	Niezmienione
`x`	x	

Napisz program `lagrange`, który wczytuje ze standardowego wejścia punkty (x_i, y_i), a następnie wypisuje na standardowe wyjście wartości wielomianu interpolacyjnego Lagrange'a w wielu gęsto leżących punktach tak, aby można było sporządzić wykres tego wielomianu w programie `gnuplot`. Dane wejściowe i wyjściowe powinny być zapisane w dwóch kolumnach, z których pierwsza zawiera odcięte, a druga rzędne, na przykład:

```
 0 15
 7  9
12 -2
```

Liczba wejściowych punktów nie jest z góry znana - program powinien odczytywać je dopóki to możliwe. Odcięte punktów wyjściowych powinny pokrywać równomiernie przedział pomiędzy najmniejszą a największą odciętą punktów wejściowych. Dla przykładowych danych wejściowych sporządź wykres wielomianu interpolacyjnego w programie `gnuplot`.

```
#include <iostream>
#include <vector>

using namespace std;

double lagrange (const vector <pair <double,double> > &points,double x) {
  double sum=0;
  for (int i=0;i!=points.size ();++i) {
    double product=1;
    for (int j=0;j!=points.size ();++j)
      if (j!=i)
        product*=(x-points [j].first)/(points [i].first-points [j].first);
    sum+=points [i].second*product; }
  return sum; }

int main () {
  vector <pair <double,double> > points;
  double x,y;
  while (cin>>x>>y)
    points.push_back (pair <double,double> (x,y));
  sort (points.begin (),points.end ());
  for (int i=0;i!=101;++i) {
    double x=points.front ().first+i*(points.back ().first-points.front ().first)/100;
    cout<<x<<" "<<lagrange (points,x)<<endl; }
  return 0; }
```

5.1.5 Figures

Dana jest lista figur z przykładu `Figure4` w następującym formacie:

```
circle red 5
rectangle blue 2 3
```

Napisz program `figures`, który wczytuje tę listę ze standardowego wejścia, a następnie wypisuje na standardowe wyjście tę samą listę posortowaną w kolejności rosnącego pola powierzchni. Przy każdej figurze powinno być dodatkowo wypisane jej pole.

```
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

class Figure {
  const string color;
  virtual ostream &print (ostream &stream) const=0;
public:
  Figure (const string &color);
  const string &getColor () const;
  virtual double getArea () const=0;
  friend ostream &operator << (ostream &stream,const Figure &figure); };

Figure::Figure (const string &color): color (color) {}

const string &Figure::getColor () const {
  return color; }

ostream &operator << (ostream &stream,const Figure &figure) {
  return figure.print (stream); }

class Circle: public Figure {
  const double r;
  ostream &print (ostream &stream) const;
public:
  Circle (const string &color,double r);
  double getArea () const;
  double getRadius () const; };

ostream &Circle::print (ostream &stream) const {
  return stream<<"circle "<<getColor ()<<" "<<r; }

Circle::Circle (const string &color,double r): Figure (color),r (r) {}

double Circle::getArea () const {
  return M_PI*r*r; }

double Circle::getRadius () const {
  return r; }

class Rectangle: public Figure {
```

```
  const double a,b;
  ostream &print (ostream &stream) const;
public:
  Rectangle (const string &color,double a,double b);
  double getArea () const;
  double getFirstEdge () const;
  double getSecondEdge () const; };

ostream &Rectangle::print (ostream &stream) const {
  return stream<<"rectangle "<<getColor ()<<" "<<a<<" "<<b; }

Rectangle::Rectangle (const string &color,double a,double b):
  Figure (color),a (a),b (b) {}

double Rectangle::getArea () const {
  return a*b; }

double Rectangle::getFirstEdge () const {
  return a; }

double Rectangle::getSecondEdge () const {
  return b; }

Figure *scan (istream &stream) {
  string name,color;
  stream>>name>>color;
  if (name=="circle") {
    double r;
    stream>>r;
    return new Circle (color,r); }
  if (name=="rectangle") {
    double a,b;
    stream>>a>>b;
    return new Rectangle (color,a,b); }
  return 0; }

bool compare (Figure *first,Figure *second) {
  return first->getArea ()<second->getArea (); }

int main () {
  vector <Figure*> figures;
  Figure *figure;
  while (figure=scan (cin))
    figures.push_back (figure);
  sort (figures.begin (),figures.end (),compare);
  for (vector <Figure*>::iterator i=figures.begin ();i!=figures.end ();++i)
    cout<<**i<<" "<<(*i)->getArea ()<<endl;
  for (vector <Figure*>::iterator i=figures.begin ();i!=figures.end ();++i)
    delete *i;
```

```
  return 0; }
```

5.1.6 Colloquium

Pewien plik tekstowy zawiera wyniki kolokwium zapisane w następującym formacie:

```
Albert Einstein 1.5 -5 0.75
Fryderyk Chopin 20.5 20 5.25
Maria Sklodowska-Curie 50 50 50
```

W każdej linii znajduje się jedno imię, jedno nazwisko, a następnie liczby punktów za kolejne zadania. Liczba zadań ani osób nie jest z góry znana, wiadomo jednak, że liczba zadań jest dla wszyskich taka sma.

Napisz program `colloquium`, który na podstawie tego pliku obliczy i wypisze całkowitą liczbę punktów zdobytych przez każdą osobę oraz średnią liczbę punktów z każdego zadania. W podanym przypadku wynikiem działania programu powinien być wydruk:

```
Albert Einstein -2.75
Fryderyk Chopin 45.75
Maria Sklodowska-Curie 150
1 24
2 21.6667
3 18.6667
```

Użycia ilu wektorów wymaga to zadanie?

```
#include <iostream>
#include <sstream>
#include <vector>

using namespace std;

int main () {
  string line;
  if (!getline (cin,line))
    return 0;
  istringstream stream (line);
  string name;
  stream>>name;
  cout<<name;
  stream>>name;
  cout<<' '<<name;
  double points;
  double sum=0;
  vector <double> sums;
  while (stream>>points) {
    sum+=points;
    sums.push_back (points); }
  cout<<' '<<sum<<endl;
  int persons;
  for (persons=1;getline (cin,line);++persons) {
```

```
    stream.clear ();
    stream.str (line);
    stream>>name;
    cout<<name;
    stream>>name;
    cout<<' '<<name;
    sum=0;
    for (int problem=0;problem<sums.size ();++problem) {
      stream>>points;
      sum+=points;
      sums [problem]+=points; }
    cout<<' '<<sum<<endl; }
  cout<<endl;
  for (int problem=0;problem<sums.size ();++problem)
    cout<<problem+1<<' '<<sums [problem]/persons<<endl;
  return 0; }
```

5.1.7 Poly

Napisz klasę `Poly` reprezentującą wielomian. Zaimplementuj:

- Funkcję składową `degree` zwracającą stopień wielomianu.
- Operator indeksowania umożliwiający odczyt i zmianę współczynnika przy zadanej potędze. Załóż, że argument jest niewiększy od stopnia wielomianu.
- Operator wywołania funkcji zwracający wartość wielomianu w zadanym punkcie.
- Operatory dodawania i mnożenia wielomianów.
- Operator `<<` wypisujący wielomian do strumienia typu `ostream` oraz operator `>>` wczytujący wielomian ze strumienia typu `istream`. Przyjmij, że wielomian zapisuje się jako sekwencję współczynników przy coraz to wyższych potęgach. Współczynniki są oddzielone przecinkami i ujęte w nawiasy. Na przykład $2 + x - 3x^2$ to `(2,1,-3)`.

Napisz program `poly`, który wczytuje ze standardowego wejścia dwa wielomiany, a następnie wypisuje na standardowe wyjście ich sumę oraz iloczyn.

```
#include <iostream>
#include <vector>

using namespace std;

class Poly {
public:
  int degree () const;
  const double &operator [] (int index) const;
  double &operator [] (int index);
  double operator () (double x) const;
  friend Poly operator + (const Poly &first,const Poly &second);
  friend Poly operator * (const Poly &first,const Poly &second);
  friend ostream &operator << (ostream &stream,const Poly &poly);
```

```
  friend istream &operator >> (istream &stream,Poly &poly);
private:
  vector <double> coff; };

int Poly::degree () const {
  return coff.size ()-1; }

const double &Poly::operator [] (int index) const {
  return coff [index]; }

double &Poly::operator [] (int index) {
  return coff [index]; }

double Poly::operator () (double x) const {
  double value=0;
  for (int index=degree ();index>=0;index--)
    value=x*value+coff [index];
  return value; }

Poly operator + (const Poly &first,const Poly &second) {
  const Poly *min,*max;
  if (first.degree ()<second.degree ()) {
    min=&first; max=&second; }
  else {
    min=&second; max=&first; }
  Poly sum=*max;
  for (int index=0;index<min->coff.size ();index++)
    sum.coff [index]+=min->coff [index];
  return sum; }

Poly operator * (const Poly &first,const Poly &second) {
  Poly product;
  product.coff.resize (first.coff.size ()+second.coff.size ()-1);
  product.coff.assign (product.coff.size (),0);
  for (int i=0;i<first.coff.size ();i++)
    for (int j=0;j<second.coff.size ();j++)
      product.coff [i+j]+=first.coff [i]*second.coff [j];
  return product; }

ostream &operator << (ostream &stream,const Poly &poly) {
  cout<<'(';
  for (int index=0;index<poly.degree ();++index)
    cout<<poly.coff [index]<<',';
  return cout<<poly.coff [poly.degree ()]<<')'; }

istream &operator >> (istream &stream,Poly &poly) {
  poly.coff.clear ();
  char character;
  double coefficient;
```

```
  cin>>character;
  do {
    cin>>coefficient>>character;
    poly.coff.push_back (coefficient); }
  while (character!=')');
  return stream; }

int main () {
  Poly p,q;
  cin>>p>>q;
  cout<<p+q<<' '<<p*q<<endl;
  return 0; }
```

5.1.8 Hermite

Wielomiany Hermite'a zadane są następującą zależnością rekurencyjną:

$$H_0(x) = 1 \qquad H_1(x) = 2x \qquad H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x)$$

Korzystając z klasy `Poly` napisz program `hermite`, który wypisuje na standardowe wyjście 10 pierwszych wielomianów Hermite'a w formacie opisanym w zadaniu `Poly`.

Wskazówka. Dodaj do klasy `Poly` konstruktory umożliwiające tworzenie dowolnych wielomianów stopnia zerowego i pierwszego.

```
#include <iostream>
#include <vector>

using namespace std;

class Poly {
  vector <double> coff;
public:
  Poly ();
  Poly (double c0);
  Poly (double c0,double c1);
  int degree () const;
  const double &operator [] (int index) const;
  double &operator [] (int index);
  double operator () (double x) const;
  friend Poly operator + (const Poly &first,const Poly &second);
  friend Poly operator * (const Poly &first,const Poly &second);
  friend ostream &operator << (ostream &stream,const Poly &poly);
  friend istream &operator >> (istream &stream,Poly &poly); };

Poly::Poly (): coff (1) {}

Poly::Poly (double c0): coff (1) {
  coff [0]=c0; }

Poly::Poly (double c0,double c1): coff (2) {
  coff [1]=c1;
```

```
  coff [0]=c0; }

int Poly::degree () const {
  return coff.size ()-1; }

const double &Poly::operator [] (int index) const {
  return coff [index]; }

double &Poly::operator [] (int index) {
  return coff [index]; }

double Poly::operator () (double x) const {
  double value=0;
  for (int index=degree ();index>=0;index--)
    value=x*value+coff [index];
  return value; }

Poly operator + (const Poly &first,const Poly &second) {
  const Poly *min,*max;
  if (first.degree ()<second.degree ()) {
    min=&first; max=&second; }
  else {
    min=&second; max=&first; }
  Poly sum=*max;
  for (int index=0;index<min->coff.size ();index++)
    sum.coff [index]+=min->coff [index];
  return sum; }

Poly operator * (const Poly &first,const Poly &second) {
  Poly product;
  product.coff.resize (first.coff.size ()+second.coff.size ()-1);
  product.coff.assign (product.coff.size (),0);
  for (int i=0;i<first.coff.size ();i++)
    for (int j=0;j<second.coff.size ();j++)
      product.coff [i+j]+=first.coff [i]*second.coff [j];
  return product; }

ostream &operator << (ostream &stream,const Poly &poly) {
  cout<<'(';
  for (int index=0;index<poly.degree ();++index)
    cout<<poly.coff [index]<<',';
  return cout<<poly.coff [poly.degree ()]<<')'; }

istream &operator >> (istream &stream,Poly &poly) {
  poly.coff.clear ();
  char character;
  double coefficient;
  cin>>character;
  do {
```

```
    cin>>coefficient>>character;
    poly.coff.push_back (coefficient); }
  while (character!=')');
  return stream; }

int main () {
  vector <Poly> hermites (10);
  hermites [0]=Poly (1);
  hermites [1]=Poly (0,2);
  for (int i=2;i<10;i++)
    hermites [i]=Poly (0,2)*hermites [i-1]+Poly (-2*(i-1))*hermites [i-2];
  for (int i=0;i<10;i++)
    cout<<hermites [i]<<endl;
  return 0; }
```

5.1.9 Histogram

Napisz klasę `Histogram` reprezentującą jednowymiarowy histogram o binach jednakowej szerokości. Zaimplementuj:

- Konstruktor pozwalający na zadanie krańców przedziału histogramowania i liczby binów oraz zerujący liczby zliczeń we wszystkich binach.
- Funkcję składową `reset` zerującą liczby zliczeń we wszystkich binach.
- Funkcję składową `insert` zwiększającą o jeden liczbę zliczeń w binie, w którym wypada liczba rzeczywista zadana zargumentem funkcji.
- Funkcję składową `count`, która wywołana bez argumentu zwraca całkowitą liczbę zliczeń w histogramie, a wywołana z argumentem całkowitym zwraca liczbę zliczeń w binie o zadanym numerze.
- Funkcję składową `print` wypisującą liczby zliczeń we wszystkich binach do zadanego strumienia wyjściowego i zwracającą referencję tego strumienia. Dane kolejnych binów powinny znajdować się w kolejnych wierszach, a każdy wiersz powinien zawierać numer binu oraz odpowiadającą mu liczbę zliczeń.

Napisz program `histogram` losujący 100 000 liczb rzeczywistych, z których każda jest sumą dwóch liczb losowych z rozkładu płaskiego w przedziale od 0 do 1. Sporządź histrogram tej próby losowej dobierając przedział histogramowania oraz liczbę binów tak, aby jak najlepiej oddać kształt otrzymanego rozkładu prawdopodobieństwa. Uzyskany histogram wydrukuj na standardowe wyjście, a następnie wykreśl go przy pomocy programu `gnuplot` pisząc na przykład

```
plot "histogram.txt" with boxes
```

```
#include <cstdlib>
#include <iostream>
#include <ostream>
#include <vector>

using namespace std;
```

```
class Histogram {
  const double lower,upper;
  vector <int> counts;
public:
  Histogram (double lower,double upper,int number);
  void reset ();
  void insert (double value);
  int count () const;
  int count (int bin) const;
  ostream &print (ostream &stream) const; };

Histogram::Histogram (double lower,double upper,int number):
  lower (lower),upper (upper),counts (number) {}

void Histogram::reset () {
  counts=vector <int> (counts.size ()); }

void Histogram::insert (double value) {
  if (lower<=value && value<upper)
    ++counts [counts.size ()*(value-lower)/(upper-lower)]; }

int Histogram::count () const {
  int sum=0;
  for (int i=0;i!=counts.size ();++i)
    sum+=counts [i];
  return sum; }

int Histogram::count (int bin) const {
  return counts [bin]; }

ostream &Histogram::print (ostream &stream) const {
  for (int i=0;i!=counts.size ();++i)
    stream<<i<<" "<<counts [i]<<endl;
  return stream; }

int main () {
  Histogram histogram (0.,2.,100);
  for (int i=0;i!=100000;++i)
    histogram.insert ((double) random ()/RAND_MAX+(double) random ()/RAND_MAX);
  histogram.print (cout);
  return 0; }
```

5.1.10 People

Każda linia pewnego pliku zawiera następujące dane osobowe:

`Imię Nazwisko Rok_Urodzenia`

zapisane w takiej właśnie kolejności i oddzielone spacjami. Imiona i nazwiska są jednoczłonowe, a liczba linii w pliku jest nieznana. Napisz program `people`, który z argumentu wywołania pobiera nazwę pliku, wczytuje ten plik do pamięci, a następnie wczytuje ze standardowego

wejścia nazwisko i wypisuje na standardowe wyjście dane wszystkich osób o tym nazwisku, w formacie takim jak w pliku z danymi.

5.1.11 ODE

Zadeklaruj abstrakcyjną klasę `Equation` reprezentującą układ równań różniczkowych postaci

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = \vec{f}(\vec{x}, t)$$

Powinna ona zawierać jedynie wirtualny operator

```
vector <double> Equation::operator () (const vector <double> &x,double t) const;
```

który na podstawie zmiennych $\vec{x}$ oraz t oblicza i zwraca wartość funkcji $\vec{f}$.

Równania różniczkowe zwyczajne rozwiązujemy numerycznie wykonując kroki czasowe skończonej długości. Niech $\vec{x}_n = \vec{x}(t_n)$. Wykonanie pojedynczego kroku o długości h polega na obliczeniu $\vec{x}_{n+1} \approx \vec{x}(t_{n+1})$, gdzie $t_{n+1} = t_n + h$. Do wykonania takiego kroku można posłużyć się różnymi metodami.

Zadeklaruj abstrakcyjną klasę `Solver`, która reprezentuje dowolną metodę wykonania pojedynczego kroku czasowego. Umieść w tej klasie:

- Konstruktor

  ```
  Solver::Solver (const Equation &equation,double step);
  ```

  umożliwiający zadanie układu do rozwiązania oraz kroku czasowego.

- Wirtualny operator

  ```
  void Solver::operator () (vector <double> &x,double &t);
  ```

  wykonujący pojedynczy krok. Na wejściu argumenty `x` oraz `t` zawierają odpowiednio $\vec{x}_n$ oraz t_n. Zadaniem operatora jest wpisanie do tych zmiennych wartości $\vec{x}_{n+1}$ oraz t_{n+1}.

Jedną z najpopularniejszych metod numerycznego rozwiązywania równań różniczkoych zwyczajnych jest schemat Rungego-Kutty czwartego rzędu. W podejściu tym dla zadanych $\vec{x}_n$ oraz t_n znajdujemy kolejno

$$\begin{aligned}
\vec{f_1} &= \vec{f}(\vec{x}_n, t_n) \\
\vec{f_2} &= \vec{f}(\vec{x}_n + h\vec{f_1}/2, t_n + h/2) \\
\vec{f_3} &= \vec{f}(\vec{x}_n + h\vec{f_2}/2, t_n + h/2) \\
\vec{f_4} &= \vec{f}(\vec{x}_n + h\vec{f_3}, t_n + h)
\end{aligned}$$

Wtedy

$$\vec{x}_{n+1} = \vec{x}_n + h(\vec{f_1}/6 + \vec{f_2}/3 + \vec{f_3}/3 + \vec{f_4}/6)$$

Z klasy `Solver` wyprowadź klasę `RK4` reprezentującą metodę Rungego-Kutty. Zaimplementuj jej wirtualne metody.

Deklaracje klas `Equation`, `Solver` oraz `RK4` umieść w pliku nagłówkowym `ode.h`, zaś definicje ich metod w pliku `ode.cpp`. Uzyskasz w ten sposób prostą bibliotekę, którą można łatwo dołączyć do każdego programu wymagającego rozwiązywania równań różniczkowych zwyczajnych.

- Plik `ode.h`

```
#ifndef ODE_H
#define ODE_H

#include <vector>

using namespace std;

class Equation {
public:
  virtual vector <double> operator () (const vector <double> &x,
                                       double t) const=0; };

class Solver {
protected:
  const Equation &equation;
  const double step;
public:
  Solver (const Equation &equation,double step);
  virtual void operator () (vector <double> &x,double &t) const=0; };

class RK4: public Solver {
public:
  RK4 (const Equation &equation,double step);
  virtual void operator () (vector <double> &x,double &t) const; };

#endif
```

- Plik `ode.cpp`

```
#include <vector>
#include "ode.h"

using namespace std;

Solver::Solver (const Equation &equation,double step):
  equation (equation),step (step) {}

RK4::RK4 (const Equation &equation,double step): Solver (equation,step) {}

void RK4::operator () (vector <double> &x,double &t) const {
  vector <double> f1=equation (x,t);
  vector <double> a (x.size ());
  for (int i=0;i<x.size ();i++)
    a [i]=x [i]+step*f1 [i]/2;
  vector <double> f2=equation (a,t+step/2);
  for (int i=0;i<x.size ();i++)
    a [i]=x [i]+step*f2 [i]/2;
  vector <double> f3=equation (a,t+step/2);
```

```
for (int i=0;i<x.size ();i++)
  a [i]=x [i]+step*f3 [i];
vector <double> f4=equation (a,t+step);
for (int i=0;i<x.size ();i++)
  x [i]+=step*(f1 [i]/6+f2 [i]/3+f3 [i]/3+f4 [i]/6);
t+=step; }
```

5.1.12 Bomb

Przyjmij, że podczas wybuchu bomby atomowej zachodzą trzy procesy:

- W wyniku zderzenia neutronu z jądrem materiału rozszczepialnego neutron jest pochłaniany, a jądro rozszczepia się na dwa fragmenty, które emitują w sumie trzy neutrony i dalej się nie rozszczepiają.
- Neutron jest pochłaniany przez fragmenty rozszczepienia i nie wywołuje dalszych rozszczepień.
- Neutron wylatuje z bomby i również nie powoduje dalszych rozszczepień.

Przy takich założeniach ilości jąder rozszczepialnych u, fragmentów rozszczepienia g oraz neutronów n spełniają równania

$$\begin{aligned} \frac{\mathrm{d}u}{\mathrm{d}t} &= -\alpha u n \\ \frac{\mathrm{d}g}{\mathrm{d}t} &= 2\alpha u n \\ \frac{\mathrm{d}n}{\mathrm{d}t} &= 2\alpha u n - \beta g n - \gamma n \end{aligned}$$

gdzie α, β, γ są stałymi. Przyjmij, że $\alpha = 10^{26}$, $\beta = 5\times10^{-27}$, $\gamma = 1$, oraz że w chwili początkowej $u = 10^{26}$, $g = 0$, $n = 10^3$.

Z klasy `Equation` wyprowadź klasę `Bomb` reprezentującą powyższy układ równań. Zdefiniuj w niej konstruktor umożliwiający zadanie parametrów α, β i γ. Napisz program `bomb`, który rozwiązuje ten układ metodą Rungego-Kutty czwartego rzędu. Dobierz krok całkowania oraz przedział czasu, w którym znajdowane jest rozwiązanie, w taki sposób, aby zaobserwować wybuch. Sporządź wykresy zależności u, g i n od czasu.

Przyjmijmy, że program znajduje się w pliku `bomb.cpp`. Program ten należy połączyć z procedurami rozwiązywania równań zawartymi w plikach `ode.h` oraz `ode.cpp`. W tym celu należy wszystkie trzy pliki umieścić w jednym katalogu i wydać w nim polecenie

```
g++ -o bomb bomb.cpp ode.cpp
```

```
#include <iostream>
#include <iomanip>
#include <vector>
#include "ode.h"

using namespace std;

class Bomb: public Equation {
  const double alpha,beta,gamma;
public:
```

```
  Bomb (double alpha,double beta,double gamma);
  virtual vector <double> operator () (const vector <double> &x,double t) const; };

Bomb::Bomb (double alpha,double beta,double gamma):
  alpha (alpha),beta (beta),gamma (gamma) {}

vector <double> Bomb::operator () (const vector <double> &x,double t) const {
  vector <double> f (3);
  f [0]= -alpha*x [0]*x [2];
  f [1]=2*alpha*x [0]*x [2];
  f [2]=2*alpha*x [0]*x [2]-beta*x [1]*x [2]-gamma*x [2];
  return f; }

int main () {
  vector <double> x (3);
  x [0]=1e26;
  x [1]=0;
  x [2]=1e3;
  double t=0;
  Bomb bomb (1e-26,5e-27,1);
  RK4 rk4 (bomb,0.5);
  cout<<scientific<<setprecision (5);
  while (t<100) {
    cout<<t<<" "<<x [0]<<" "<<x [1]<<" "<<x [2]<<endl;
    rk4 (x,t); }
  return 0; }
```

5.1.13 Arenstorf

Równanie różniczkowe zwyczajne drugiego rzędu postaci

$$\frac{\mathrm{d}^2\vec{u}}{\mathrm{d}t^2} = \vec{g}\left(\frac{\mathrm{d}\vec{u}}{\mathrm{d}t}, \vec{u}, t\right)$$

można sprowadzić do równania pierwszego rzędu wprowadzając zmienne

$$\vec{v} = \frac{\mathrm{d}\vec{u}}{\mathrm{d}t}$$

Wyjściowe równanie staje się bowiem równoważne następującemu układowi z dwukrotnie większą liczbą niewiadomych:

$$\begin{aligned} \frac{\mathrm{d}\vec{u}}{\mathrm{d}t} &= \vec{v} \\ \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} &= \vec{g}(\vec{v}, \vec{u}, t) \end{aligned}$$

Układ ten rozwiązujemy traktując $\vec{u}$ i $\vec{v}$ jako niezależne zmienne, gdyż spełnienie warunku $\vec{v} = \mathrm{d}\vec{u}/\mathrm{d}t$ jest zapewnione przez górne równanie. Znane nam procedury można zatem wykorzystać również do rozwiązywania równań drugiego rzędu biorąc $\vec{x}$ i $\vec{f}$ w postaci

$$\vec{x} = \begin{bmatrix} \vec{u} \\ \vec{v} \end{bmatrix} \qquad \vec{f}(\vec{x}, t) = \begin{bmatrix} \vec{v} \\ \vec{g}(\vec{v}, \vec{u}, t) \end{bmatrix}$$

Rozważ ruch lekkiego satelity w płaszczyznie obrotu izolowanego układu planeta - księżyc. Księżyc ma masę μ, zaś planeta $1-\mu$. W nieinercjalnym układzie odniesienia, w którym planeta znajduje się w punkcie $(-\mu, 0)$, a księżyc w punkcie $(1-\mu, 0)$, ruch satelity opisują równania

$$\begin{aligned}\frac{\mathrm{d}^2x}{\mathrm{d}t^2} &= x + 2\frac{\mathrm{d}y}{\mathrm{d}t} - (1-\mu)\frac{x+\mu}{d_1} - \mu\frac{x-1+\mu}{d_2} \\ \frac{\mathrm{d}^2y}{\mathrm{d}t^2} &= y - 2\frac{\mathrm{d}x}{\mathrm{d}t} - (1-\mu)\frac{y}{d_1} - \mu\frac{y}{d_2}\end{aligned}$$

gdzie

$$d_1 = \left((x+\mu)^2 + y^2\right)^{3/2} \qquad d_2 = \left((x-1+\mu)^2 + y^2\right)^{3/2}$$

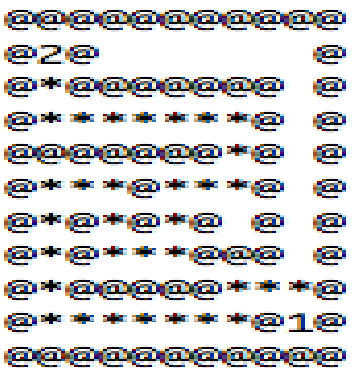
Okazuje się, że dla odpowiednio małych μ istnieją orbity zamknięte o dość wymyślych kształtach, zwane orbitami Arenstorfa. Jedną z nich otrzymuje się dla $\mu = 0.012277471$ i następujących warunków początkowych:

$$x = 0.994 \quad \frac{\mathrm{d}x}{\mathrm{d}t} = 0 \quad y = 0 \quad \frac{\mathrm{d}y}{\mathrm{d}t} = -2.0015851063790825$$

Napisz program `arenstorf`, który znajduje tę orbitę metodą Rungego-Kutty czwartego rzędu. Weź pod uwagę, że dobranie odpowiedniego kroku oraz znalezienie okresu obiegu jest w tym przypadku dosyć trudne. Sporządź wykres otrzymanej orbity.

5.1.14 Labyrinth

Napisz program `labyrinth` znajdujący drogę między dwoma punktami w labiryncie. Plansza labiryntu zapisana jest w pliku tekstowym w formacie takim, jak na rysunku:



Symbol `@` oznacza mur. Po labiryncie można chodzić tylko poziomo lub pionowo. Cały labirynt jest otoczony murem, to znaczy algorytmowi nie grozi wyjście poza planszę. Zadaniem programu jest znalezienie drogi pomiędzy punktami oznaczonymi na planszy cyframi `1` i `2`. Program powinien wczytać planszę ze standardowego wejścia, a następnie wypisać na standardowe wyjście planszę z drogą zaznaczoną gwiazdkami, jak na rysunku.

Wskazówka. Planszę labiryntu wygodnie jest pamiętać w wektorze łańcuchów, czyli w strukturze typu `vector <string>`.

```
#include <iostream>
#include <string>
#include <vector>
```

```
using namespace std;

vector <string> lab;

bool go (int r,int c) {
  if (lab [r][c]=='2')
    return true;
  lab [r][c]='#';
  for (int v=-1;v<=1;v++)
    for (int h=-1;h<=1;h++) {
      if ((v && h) || !(v || h))
        continue;
      if (lab [r+v][c+h]!='@' && lab [r+v][c+h]!='#')
        if (go (r+v,c+h)) {
          lab [r][c]='*';
          return true; }}
  return false; }

int main () {
  string row;
  while (getline (cin,row))
    lab.push_back (row);
  int r,c;
  for (r=0;r<lab.size ();r++) {
    for (c=0;c<lab [r].size ();c++)
      if (lab [r][c]=='1')
        break;
    if (lab [r][c]=='1')
      break; }
  if (!go (r,c)) {
    cout<<"Nie ma przejscia"<<endl;
    return 1; }
  lab [r][c]='1';
  for (r=0;r<lab.size ();r++)
    for (c=0;c<lab [r].size ();c++)
      if (lab [r][c]=='#')
        lab [r][c]=' ';
  for (int r=0;r<lab.size ();r++)
    cout<<lab [r]<<endl;
  return 0; }
```

5.2 Set

5.2.1 Set - przykład

Poniższy przykład ilustruje wykorzystanie pojemnika `set` z biblioteki STL.

```
#include <iostream>
#include <set>
```

```
using namespace std;

int main () {
  set <char> s;
  s.insert ('k');
  s.insert ('b');
  s.insert ('k');
  cout<<s.size ()<<endl;
  cout<<s.count ('a')<<" "<<s.count ('b')<<" "<<s.count ('k')<<endl;
  for (set <char>::iterator i=s.begin ();i!=s.end ();++i)
    cout<<*i<<endl;
  return 0; }
```

Wykonanie tego programu daje taki wydruk:

```
2
0 1 1
b
k
```

5.2.2 Exclusive

Napisz program `exclusive` znajdujący liczbę parami różnych słów w zadanym pliku tekstowym.

```
#include <iostream>
#include <set>

using namespace std;

int main () {
  set <string> words;
  string word;
  while (cin>>word)
    words.insert (word);
  cout<<words.size ()<<endl;
  return 0; }
```

5.2.3 Lotto

Napisz program `lotto` losujący 6 parami różnych liczb naturalnych z przedziału od 1 do 49 włącznie i wypisujący je na standardowe wyjście w kolejności rosnącej. W programie nie wolno wołać procedury sortującej.

```
#include <iostream>
#include <set>
#include <cstdlib>
#include <ctime>

using namespace std;

int main () {
```

```
  srand (time (0));
  set <int> lotto;
  do
    lotto.insert (1+49.*rand ()/(1.+RAND_MAX));
  while (lotto.size ()!=6);
  for (set <int>::iterator iter=lotto.begin ();iter!=lotto.end ();++iter)
    cout<<*iter<<endl;
  return 0; }
```

5.2.4 Login

W pewnym serwisie internetowym każdy użytkownik identyfikowany jest przy pomocy numeru z przedziału od 0 do 999999. Numery zarejestrowanych użytkowników są przechowywane w pliku `login.txt`, jeden numer w każdej linii. Napisz program `login` umożliwiający logowanie się użytkowników do serwisu. Program powinien:

1. Wypisywać komunikat zachęcający do podania numeru lub wpisania słowa `new`.
2. W przypadku podania numeru wypisać informację, czy logowanie się powiodło, w zależności od tego, czy dany numer znajduje się w pliku `login.txt`.
3. W przypadku wpisania `new` przydzielić użytkownikowi losowo nowy numer, wypisać go i dodać do pliku `login.txt`.

```
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <set>

using namespace std;

int main () {
  int id;
  set <unsigned int> ids;
  ifstream input ("login.txt");
  while (input>>id)
    ids.insert (id);
  input.close ();
  cout<<"Enter your ID or type \'new\' to obtain a new one: ";
  if (cin>>id)
    cout<<(ids.count (id) ? "Login successful." : "Unknown user.")<<endl;
  else {
    while (!ids.insert (id=random ()%1000000).second);
    cout<<"Your ID is "<<id<<"."<<endl; }
  ofstream output ("login.txt");
  for (set <unsigned int>::iterator it=ids.begin ();it!=ids.end ();++it)
    output<<*it<<endl;
  output.close ();
  return 0; }
```

5.2.5 Intersection

Napisz program `intersection`, który wczytuje ze standardowego wejścia dwie linie liczb całkowitych, a następnie wypisuje na stadnardowe wyjście te liczby, które występują w obu liniach jednocześnie. Liczby powinny być wypisywane w kolejności rosnącej i każda tylko raz. W programie nie wolno wołać procedury sortującej.

```
#include <iostream>
#include <sstream>
#include <string>
#include <set>

using namespace std;

void intersection (set <int> &s,const set <int> &s1,const set <int> &s2) {
  s.clear ();
  set <int>::iterator i1=s1.begin (),i2=s2.begin ();
  while (i1!=s1.end () && i2!=s2.end ()) {
    if (*i1==*i2)
      s.insert (*i1);
    ++(*i1<*i2 ? i1 : i2); }}

int main () {
  set <int> s1,s2,s;
  string line;
  getline (cin,line);
  istringstream stream (line);
  int number;
  while (stream>>number)
    s1.insert (number);
  getline (cin,line);
  stream.str (line);
  stream.clear ();
  while (stream>>number)
    s2.insert (number);
  intersection (s,s1,s2);
  for (set <int>::iterator i=s.begin ();i!=s.end ();++i)
    cout<<*i<<" ";
  cout<<endl;
  return 0; }
```

5.2.6 Library

Napisz klasę `Book` pamiętającą autorów i tytuł książki. Lista autorów może być pojedynczą zmienną łańcuchową zawierającą imiona i nazwiska autorów w dowolnej kolejności. Zaimplementuj:

- Funkcję składową `contains` sprawdzającą, czy lista autorów lub tytuł zawierają zadany łańcuch znaków.
- Operator `<` określający porządek książek wyznaczony kolejnością alfabetyczną list autorów a następnie tytułów.

- Operator `>>` wczytujący dane książki ze strumienia typu `istream` oraz operator `<<` wypisujący dane książki do strumienia typu `ostream`. Dane książki zapisujemy w dwóch bezpośrednio po sobie następujących liniach zawierających kolejno listę autorów oraz tytuł.

Napisz klasę `Library` reprezentującą spis książek. Zaimplementuj:

- Funkcję składową `find`, która zwraca obiekt typu `Library` ze spisem tych książek, których nazwa autora lub tytuł zawierają zadany łańcuch znaków.
- Operator `>>` wczytujący spis książek ze strumienia typu `istream` oraz operator `<<` wypisujący spis książek do strumienia typu `ostream`. Spis książek zapisujemy w pliku jak w poniższym przykładzie:

```
andrzej kajetan wroblewski
historia fizyki
stanislaw szpikowski
elementy mechaniki kwantowej
andrzej kajetan wroblewski, janusz andrzej zakrzewski
wstep do fizyki
```

Napisz program `library` pozwalający na wyszukiwanie książek w spisie. Program wczytuje spis z pliku, którego nazwa podana jest jako argument wywołania programu. Następnie czyta ze standardowego wejścia jedną linię tekstu i wypisuje na standardowe wyjście dane tych książek, których lista autorów lub tytuł zawierają sekwencję znaków zawartą we wprowadzonej linii.

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

using namespace std;

struct Book {
  string author,title;
  bool contains (const string &keyword) const;
  friend bool operator < (const Book &first,const Book &second);
  friend istream &operator >> (istream &stream,Book &book);
  friend ostream &operator << (ostream &stream,const Book &book); };

bool Book::contains (const string &keyword) const {
  return author.find (keyword)!=string::npos || title.find (keyword)!=string::npos; }

bool operator < (const Book &first,const Book &second) {
  return first.author<second.author || (first.author==second.author && first.title<second.ti

istream &operator >> (istream &stream,Book &book) {
  getline (stream,book.author);
  getline (stream,book.title);
  return stream; }

ostream &operator << (ostream &stream,const Book &book) {
```

```
  return stream<<book.author<<endl<<book.title<<endl; }

class Library {
  set <Book> books;
public:
  Library find (const string &keyword);
  friend istream &operator >> (istream &stream,Library &library);
  friend ostream &operator << (ostream &stream,const Library &library); };

Library Library::find (const string &keyword) {
  Library library;
  for (set <Book>::iterator i=books.begin ();i!=books.end ();++i)
    if (i->contains (keyword))
      library.books.insert (*i);
  return library; }

istream &operator >> (istream &stream,Library &library) {
  library.books.clear ();
  Book book;
  while (stream>>book)
    library.books.insert (book);
  return stream; }

ostream &operator << (ostream &stream,const Library &library) {
  for (set <Book>::const_iterator i=library.books.begin ();i!=library.books.end ();++i)
    stream<<*i;
  return stream; }

int main (int argc,char *argv []) {
  Library library;
  ifstream input (argv [1]);
  input>>library;
  input.close ();
  string keyword;
  while (cin>>keyword)
    cout<<library.find (keyword);
  return 0; }
```

5.2.7 Sudoku

Napisz program `sudoku` rozwiązujący standardowe sudoku $3^2 \times 3^2$.

```
#include <iostream>
#include <vector>
#include <set>

using namespace std;

class Solver {
  vector <set <short> > links;
```

```
  short first (vector <short> &initial);
  short backward (vector <short> &initial,short &index);
  short forward (vector <short> &initial,short &index);
  bool check (vector <short> &solution,short index);
  short increment (vector <short> &solution,short index);
public:
  Solver (set <set <short> > &groups,short size);
  vector <short> operator () (vector <short> &initial,short number); };

short Solver::first (vector <short> &initial) {
  short index;
  for (index=0;initial [index];++index);
  return index; }

short Solver::backward (vector <short> &initial,short &index) {
  for (--index;index>=0 && initial [index];--index);
  return index; }

short Solver::forward (vector <short> &initial,short &index) {
  for (++index;index<initial.size () && initial [index];++index);
  return index; }

bool Solver::check (vector <short> &solution,short index) {
  for (set <short>::iterator i=links [index].begin ();i!=links [index].end ();++i)
    if (solution [*i]==solution [index])
      return true;
  return false; }

short Solver::increment (vector <short> &solution,short index) {
  for (++solution [index];check (solution,index);++solution [index]);
  return solution [index]; }

Solver::Solver (set <set <short> > &groups,short size): links (size) {
  for (set <set <short> >::iterator i=groups.begin ();i!=groups.end ();++i)
    for (set <short>::iterator j=i->begin ();j!=i->end ();++j)
      for (set <short>::iterator k=i->begin ();k!=i->end ();++k)
        if (*j!=*k)
          links [*j].insert (*k); }

vector <short> Solver::operator () (vector <short> &initial,short number) {
  vector <short> solution=initial;;
  short index=first (initial);
  while (index>=0 && index<initial.size ())
    if (increment (solution,index)>number) {
      solution [index]=0;
      backward (initial,index); }
    else
      forward (initial,index);
  return index>0 ? solution : vector <short> (); }
```

```
short index (short i,short j) {
  return 9*i+j; }

int main () {
  set <set <short> > groups;
  for (int i=0;i<9;++i) {
    set <short> horizontal,vertical;
    for (int j=0;j<9;++j) {
      horizontal.insert (index (i,j));
      vertical.insert (index (j,i)); }
    groups.insert (horizontal);
    groups.insert (vertical); }
  for (int i=0;i<3;++i)
    for (int j=0;j<3;++j) {
      set <short> group;
      for (int k=0;k<3;++k)
        for (int l=0;l<3;++l)
          group.insert (index (3*i+k,3*j+l));
      groups.insert (group); }
  vector <short> initial (81);
  for (int i=0;i<81;++i)
    cin>>initial [i];
  vector <short> solution=Solver (groups,81) (initial,9);
  if (solution.size ())
    for (int i=0;i<9;++i) {
      for (int j=0;j<9;++j)
        cout<<" "<<solution [index (i,j)];
      cout<<endl; }
  return 0; }
```

5.3 Map

5.3.1 Map - przykład

Poniższy przykład ilustruje wykorzystanie pojemnika `map`.

```
#include <iostream>
#include <map>

using namespace std;

int main () {
  map <string,int> m;
  m ["ala"]=5;
  m ["ma"]=7;
  cout<<m ["ala"]<<" "<<m ["ma"]<<" "<<m ["kota"]<<endl;
  cout<<m.size ()<<endl;
  for (map <string,int>::iterator i=m.begin ();i!=m.end ();++i)
    cout<<i->first<<" "<<i->second<<endl;
  return 0; }
```

Wykonanie tego programu daje taki wydruk:

```
5 7 0
3
ala 5
kota 0
ma 7
```

5.3.2 Substring

Pewien plik tekstowy zawiera jedną linię tekstu. Napisz program `substring`, który znajduje najczęściej występującą w tym pliku sekwencję n znaków wliczając znaki białe. Długość sekwencji n powinna być zadawana jako argument wywołania programu. Program przetestuj na tekście *Pana Tadeusza* i *Hamleta*.

```
#include <iostream>
#include <sstream>
#include <string>
#include <map>

using namespace std;

int main (int argc,char *argv []) {
  int length;
  istringstream (argv [1])>>length;
  string line;
  getline (cin,line);
  map <string,int> ocurrences;
  for (int index=0;index!=line.size ()-length+1;++index)
    ++ocurrences [line.substr (index,length)];
  map <string,int>::iterator max=ocurrences.begin ();
  for (map <string,int>::iterator iter=ocurrences.begin ();
       iter!=ocurrences.end ();
       ++iter)
    if (iter->second>max->second)
      max=iter;
  cout<<max->first<<endl;
  return 0; }
```

5.3.3 Nuclide

Napisz klasę `Nuclide` reprezentującą nuklid o zadanej liczbie protonów i neutronów. Zaimplementuj:

- Statyczną funckję składową `initialize` wczytującą symbole pierwiastków chemicznych ze strumienia zadanego swoim argumentem. Przyjmij, że zawartość tego strumienia wygląda następująco:

  ```
  n H He Li Be B ... Ns Hs Mt
  ```

- Operator `<` wyznaczający kolejność nuklidów ze względu na liczbę protonów, a w przypadku jednakowej liczby protonów ze względu na liczbę neutronów.

- Operator `<<` wypisujący obiekt klasy `Nuclide` do strumienia typu `ostream` oraz operator `>>` wczytujący taki obiekt ze strumienia typu `istream`. Przyjmij, że nuklidy zapisujemy w postaci ich symbolu chemicznego poprzedzonego całkowitą liczbą nukleonów, na przykład `235U`, `239Pu`.

Odpowiedni plik z symbolami pierwiastków znajduje się pod adresem

`http://www.fuw.edu.pl/~polbrat/elements.txt`

5.3.4 Chain

Plik

`http://www.fuw.edu.pl/~polbrat/decays.txt`

wylicza możliwe kanały rozpadu wszystkich znanych jąder promieniotwórczych. Każda linia zawiera informacje o rozpadach jednego jądra. Przykładowo, linia

```
179Hg 175Pt 178Pt 179Au
```

oznacza, że ^{179}Hg może się rozpaść do ^{175}Pt, ^{178}Pt lub ^{179}Au.

Napisz program `chain`, który wczyta ze standardowego wejścia dowolną listę nuklidów i wypisze na standardowe wyjście wszystkie nuklidy, które mogą z nich powstać w wyniku wielokrotnych rozpadów promieniotwórczych.

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <map>
#include <set>

using namespace std;

class Nuclide {
  static vector <string> atomicToSymbol;
  static map <string,int> symbolToAtomic;
  int N,Z;
public:
  static void initialize (istream &stream);
  friend bool operator < (const Nuclide &first,const Nuclide &second);
  friend istream &operator >> (istream &stream,Nuclide &nuclide);
  friend ostream &operator << (ostream &stream,const Nuclide &nuclide); };

vector <string> Nuclide::atomicToSymbol=vector <string> ();
map <string,int> Nuclide::symbolToAtomic=map <string,int> ();

void Nuclide::initialize (istream &stream) {
  string symbol;
  while (stream>>symbol) {
    symbolToAtomic [symbol]=atomicToSymbol.size ();
```

```
    atomicToSymbol.push_back (symbol); }}

bool operator < (const Nuclide &first,const Nuclide &second) {
  return first.Z<second.Z || first.Z==second.Z && first.N<second.N; }

istream &operator >> (istream &stream,Nuclide &nuclide) {
  string symbol;
  if (stream>>nuclide.N>>symbol)
    nuclide.N-=(nuclide.Z=Nuclide::symbolToAtomic [symbol]);
  return stream; }

ostream &operator << (ostream &stream,const Nuclide &nuclide) {
  return stream<<nuclide.N+nuclide.Z<<Nuclide::atomicToSymbol [nuclide.Z]; }

map <Nuclide,set <Nuclide> > decays;
set <Nuclide> initial,final;

void insert (const Nuclide &nuclide) {
  if (!final.count (nuclide)) {
    final.insert (nuclide);
    for (set <Nuclide>::iterator i=decays [nuclide].begin ();
         i!=decays [nuclide].end ();
         ++i)
      insert (*i); }}

int main () {
  ifstream file ("elements.txt");
  Nuclide::initialize (file);
  file.close ();
  file.open ("decays.txt");
  string line;
  while (getline (file,line)) {
    istringstream stream (line);
    Nuclide parent;
    stream>>parent;
    Nuclide child;
    while (stream>>child)
      decays [parent].insert (child); }
  file.close ();
  Nuclide nuclide;
  while (cin>>nuclide)
    initial.insert (nuclide);
  for (set <Nuclide>::iterator i=initial.begin ();i!=initial.end ();++i)
    insert (*i);
  for (set <Nuclide>::iterator i=final.begin ();i!=final.end ();++i)
    cout<<*i<<endl;
  return 0; }
```

5.4 MultiMap

5.4.1 Multimap - przykład

Poniższy przykład ilustruje wykorzystanie pojemnika `multimap`.

```
#include <iostream>
#include <string>
#include <map>

using namespace std;

int main () {
  multimap <string,int> m;
  m.insert (pair <string,int> ("neg",-1));
  m.insert (pair <string,int> ("pos",7));
  m.insert (pair <string,int> ("neg",-2));
  cout<<m.size ()<<endl;
  cout<<m.count ("neg")<<" "<<m.count ("pos")<<" "<<m.count ("ala")<<endl;
  for (multimap <string,int>::iterator i=m.begin ();i!=m.end ();++i)
    cout<<i->first<<" "<<i->second<<endl;
  return 0; }
```

Wykonanie tego programu daje taki wydruk:

```
3
2 1 0
neg -1
neg -2
pos 7
```

5.4.2 Junk

Linuksowe polecenie `du -k` wyświetla informację o wykorzystaniu miejsca na dysku przez poszczególne pliki i katalogi. Jest ona wypisywana na standardowe wyjście w następującej postaci:

```
5004   ./Library/Caches/Google Earth/webdata/general
5004   ./Library/Caches/Google Earth/webdata
399352 ./Library/Caches/Google Earth
3148   ./Library/Caches/iMovie
```

Pierwsza kolumna zawiera rozmiar danego pliku lub katalogu w kilobajtach, a druga jego nazwę wraz ze ścieżką dostępu.

Standardowo pliki i katalogi wypisywane są w kolejności alfabetycznej ich nazw. Napisz program `junk`, który wczytuje ze standardowego wejścia listę plików i katalogów wygenerowaną poleceniem `du -k`, a następnie wypisuje na standardowe wyjście tę samą listę posortowaną według rozmiarów w kolejności malejącej. Program przetestuj pisząc

```
du -k | junk
```

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main () {
  multimap <int,string> items;
  int length;
  string name;
  while (cin>>length>>name)
    items.insert (pair <int,string> (length,name));
  for (multimap <int,string>::iterator i=items.begin ();i!=items.end ();++i)
    cout<<i->first<<" "<<i->second<<endl;
  return 0; }
```

5.4.3 Frequency

Pewien plik tekstowy zawiera tylko słowa pisane małymi literami i oddzielone znakami białymi. Napisz program `frequency`, który wczytuje ten plik, a następnie wypisuje wszystkie wyrazy występujące w tekście w kolejności liczby wystąpień, podając przy każdym wyrazie stosunek liczby jego wystąpień do całkowitej liczby słów w tekście. Program przetestuj na tekście *Pana Tadeusza* i *Hamleta*.

```
#include <iostream>
#include <string>
#include <map>

using namespace std;

int main () {
  int total;
  map <string,int> occurences;
  string word;
  for (total=0;cin>>word;total++)
    occurences [word]++;
  multimap <int,string> words;
  for (map <string,int>::iterator item=occurences.begin ();
       item!=occurences.end ();
       ++item)
    words.insert (pair <int,string> (item->second,item->first));
  for (multimap <int,string>::iterator item=words.begin ();
       item!=words.end ();
       ++item)
    cout<<item->second<<" "<<100.*item->first/total<<endl;
  return 0; }
```

5.5 Stack

5.5.1 Reversi

Napisz program `reversi`, który wczytuje ze standardowego wejścia ciąg słów, a następnie wypisuje je na standardowe wyjście w odwrotnej kolejności, oddzielone spacjami. Program przetestuj na tekście *Pana Tadeusza* i *Hamleta*.

```
#include <iostream>
#include <string>
#include <stack>

using namespace std;

int main () {
  string slowo;
  stack <string> stos;
  while (cin>>slowo)
    stos.push (slowo);
  while (!stos.empty ()) {
    cout<<stos.top ()<<" ";
    stos.pop (); }
  cout<<endl;
  return 0; }
```

5.5.2 Ariadna

Zagraniczny student przybył do Warszawy aby kształcić się na Wydziale Fizyki Uniwersytetu Warszawskiego. Pierwszego dnia zajęć wyruszył on pieszo z akademika przy Żwirki i Wigury na ćwiczenia z Programowania przy ulicy Hożej. Idąc zapisywał na przemian jaką ulicą idzie oraz w którą stronę skręca lub czy idzie prosto wchodząc w kolejną ulicę. Otrzymał następujący zapis:

```
Żwirki i Wigury
prawo
Wawelska
lewo
Niepodległości
prosto
Chałubińskiego
prawo
Hoża
```

Na ćwiczeniach postanowił napisać program, który na podstawie tego zapisu zaproponuje mu drogę powrotną, która w tym przypadku byłaby określona następująco:

```
Hoża
lewo
Chałubińskiego
prosto
Niepodległości
prawo
```

```
Wawelska
lewo
Żwirki i Wigury
```

Napisz program `ariadna`, który realizuje to zadanie dla dowolnej trasy. Przyjmij, że nazwy ulic oraz określenia kierunków zapisane są w pliku tekstowym w kolejnych liniach. Program powinien czytać drogę tam ze standardowego wejścia i wypisywać drogę powrotną na standardowe wyjście.

5.5.3 RPN

Odwrotna Notacja Polska (ONP) jest to sposób zapisu wyrażeń arytmetycznych, który umożliwia łatwe obliczenie ich wartości przez komputer. W notacji tej symbol działania zapisuje się po argumentach, na których ma być ono wykonane. Na przykład wyrażenie `2+3` ma w ONP postać `2 3 +`, natomiast wyrażenie `sin 0.5` ma postać `0.5 sin`. Pełny algorytm obliczania wartości wyrażeń w ONP przedstawia się następująco.

- Dla każdego elementu wyrażenia:
  - Jeżeli element jest liczbą:
    - Odłóż ją na stos.
  - Jeżeli element jest symbolem działania
    - Zdejmij ze stosu tyle liczb, ilu argumentów wymaga działanie, wykonaj na nich to działanie, a jego wynik odłóż na stos.

Jeżeli wyrażenie jest poprawne, to po wykonaniu tych czynności na stosie znajdzie się wartość całego wyrażenia. Argumenty działań nieprzemiennych powinno się brać w takiej kolejności, w jakiej występują one w wyrażeniu. Na przykład wartością wyrażenia `1 2 /` powinno być `1/2=0.5`.

Napisz procedurę `rpn`, która oblicza wartość wyrażenia w ONP. Deklaracja:

```
double rpn (const string &expression);
```

Argument	**Wejście**	**Wyjście**
`expression`	Wyrażenie w ONP	
`rpn`		Wartość wyrażenia

Załóż, że poszczególne liczby oraz symbole działań są oddzielone spacjami. Procedura powinna obsługiwać przynajmniej dodawanie, odejmowanie, mnożenie i dzielenie.

Napisz program `rpn`, który wczytuje ze standardowego wejścia kolejne linie i wypisuje na standardowe wyjście wartości wyrażeń zapisanych w tych liniach.

```
#include <iostream>
#include <sstream>
#include <string>
#include <stack>
#include <cmath>

using namespace std;
```

```
double rpn (string &expression) {
  stack <double> lifo;
  string item;
  istringstream stream (expression);
  while (stream>>item) {
    if (item=="add") {
      double x=lifo.top ();
      lifo.pop ();
      double y=lifo.top ();
      lifo.pop ();
      lifo.push (y+x);
      continue; }
    if (item=="sub") {
      double x=lifo.top ();
      lifo.pop ();
      double y=lifo.top ();
      lifo.pop ();
      lifo.push (y-x);
      continue; }
    if (item=="mul") {
      double x=lifo.top ();
      lifo.pop ();
      double y=lifo.top ();
      lifo.pop ();
      lifo.push (y*x);
      continue; }
    if (item=="div") {
      double x=lifo.top ();
      lifo.pop ();
      double y=lifo.top ();
      lifo.pop ();
      lifo.push (y/x);
      continue; }
    if (item=="sqrt") {
      double x=lifo.top ();
      lifo.pop ();
      lifo.push (sqrt (x));
      continue; }
    if (item=="pi") {
      lifo.push (M_PI);
      continue; }
    istringstream stream (item);
    double x;
    stream>>x;
    lifo.push (x); }
  double x=lifo.top ();
  lifo.pop ();
  return x; }
```

```
int main () {
  string expression;
  while (getline (cin,expression))
    cout<<rpn (expression)<<endl;
  return 0; }
```

5.5.4 Parser

ONP nazywa się też notacją postfiksową, ponieważ symbole działań znajdują się w niej na końcu. Analogicznie zwykła notacja nazywa się też infiksową, gdyż symbole działań znajdują się między liczbami. Aby obliczyć wartość wyrażenia infiksowego najprościej przetłumaczyć je najpierw na ONP. Czynność tę nazywa się parsowaniem. Algorytm parsowania wyrażeń infiksowych zawierających liczby, operatory dwuargumentowe oraz nawiasy, jest następujący.

- Dla każdego elementu wyrażenia infixowego:
  - Jeśli element jest liczbą:
    - Dopisz go do wyrażenia postfixowego.
  - Jeśli element jest operatorem:
    - Dopóki stos jest niepusty oraz element na stosie jest operatorem o priorytecie wyższym niż element:
      - Zdejmij operator ze stosu i dodaj go do wyrażenia postfixowego.
    - Połóż element na stosie.
  - Jeżeli element jest lewym nawiasem:
    - Odłóż go na stos.
  - Jeżeli element jest prawym nawiasem:
    - Dopóki element na stosie nie jest lewym nawiasem:
      - Zdejmij go ze stosu i dopisz do wyrażenia postfixowego.
    - Zdejmij lewy nawias ze stosu.
- Dopóki stos nie jest pusty:
  - Zdejmuj elementy ze stosu i dopisuj je do wyrażenia postfixowego.

Występuje tu pojęcie priorytetu operatora. Można skorzystać z konwencji języka `C++`, gdzie mnożenie i dzielenie mają priorytet 13, zaś dodawanie i odejmowanie - 12.

Napisz klasę `Parser` umożliwiającą parsowanie wyrażeń infiksowych zawierających liczby, symbole mnożenia, dzielenia, dodawania i odejmowania oraz nawiasy. Zaimplementuj w niej:

- Prywatną statyczną funkcję `isoperator` stwierdzającą, czy dane element wyrażenia jest operatorem. Deklaracja:

  ```
  bool Parser::isoperator (const string &element);
  ```

- Prywatną statyczną funkcję `priority` zwracającą priorytet operatora. Deklaracja:

  ```
  int Parser::priority (const string &element);
  ```

- Publiczną statyczną funkcję `parse` parsującą wyrażenie infiksowe i zwracającą wyrażenie postfiksowe. Deklaracja:

`string Parser::parse (const string &expression);`

Załóż, że poszczególne elementy wyrażenia infiksowego są oddzielone spacjami. Wynikowe wyrażenie postfiksowe powinno spełniać ten sam warunek.

Napisz program `parser`, który wczytuje ze standardowego wejścia kolejne linie zawierające wyrażenia infiksowe i wypisuje na standardowe wyjście odpowiadające im wyrażenia postfiksowe.

5.5.5 Calculator

Korzystając z procedury `rpn` oraz klasy `Parser` napisz program `calculator`, który wczytuje ze standardowego wejścia kolejne linie zawierające wyrażenia infiksowe i wypisuje na standardowe wyjście ich wartości.

5.6 Algorytmy

5.6.1 Length

Rozważmy tablice słów typu `string` składających się wyłącznie z małych liter alfabetu łacińskiego. Napisz funkcję sortującą taką tablicę rosnąco względem liczby liter w słowie. W przypadku słów jednakowej długości ich uporządkowanie wyznacza kolejność alfabetyczna.

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

bool compare (const string &first,const string &second) {
  return first.size ()<second.size () || first.size ()==second.size () && first<second; }

int main () {
  vector <string> words;
  string word;
  while (cin>>word)
    words.push_back (word);
  sort (words.begin (),words.end (),compare);
  for (int counter=0;counter<words.size ();++counter)
    cout<<words [counter]<<endl;
  return 0; }
```

6 Biblioteka Qt

6.1 Widgety, qmake

6.1.1 Editor1

Niniejszy przykład pokazuje jak utworzyć prostą aplikację okienkową Qt przy użyciu programu `qmake`. Aplikacja nazywa się `Editor` i wyświetla okienko do edycji tekstu zawierające słowa `Przykladowy tekst`.

1. Utwórz katalog o nazwie `Editor`.
2. W katalogu tym umieść plik `main.cpp` zawierający poniższy program.
3. Wydaj komendę `qmake -project`. Spowoduje ona utworzenie pliku `Editor.pro` zawierającego dane o tzw. projekcie biblioteki Qt.
4. Wydaj polecenie `qmake`. Spowoduje ona utworzeniu pliku `Makefile`.
5. Wydaj komendę `make`, która skompiluje program tworząc plik wykonywalny o nazwie `Editor`.
6. Uruchom aplikację pisząc `./Editor` lub klikając na jej ikonę w środowisku graficznym.

```
#include <QtGui>

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  QTextEdit textEdit ("Przykladowy tekst");
  textEdit.show ();
  return application.exec (); }
```

6.2 Sygnały

6.2.1 Editor2

Niniejszy przykład ilustruje działanie sygnałów i gniazd. Program wyświetla w dwóch oddzielnych okienkach pole do edycji tekstu oraz przycisk z napisem `Clear`, którego naciśnięcie powoduje wyczyszczenie pola do edycji tekstu.

```
#include <QtGui>

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  QTextEdit *textEdit=new QTextEdit ("Przykladowy tekst");
  QPushButton *pushButton=new QPushButton ("Clear");
  application.connect (pushButton,SIGNAL (clicked ()),textEdit,SLOT (clear ()));
  textEdit->show ();
  pushButton->show ();
  return application.exec (); }
```

6.3 Klasa główna, QtCreator

6.3.1 Editor3

Niniejszy przykład pokazuje, jak napisać aplikację okienkową Qt z wykorzystaniem programu QtCreator, lecz bez użycia programu QtDesigner. Ilustruje też tworzenie własnej klasy pochodnej od klasy `QWidget` oraz umieszczanie jednych widgetów w drugich. Program wyświetla w jednym okienku pole do edycji tekstu oraz przycisk `Clear`, którego naciśnięcie usuwa cały tekst.

Aby utworzyć aplikację okienkową:

1. Z głównego menu wybierz `File->New File or Project...`.
2. Z rozwijanej listy `Projects` wybierz `Qt4 Gui Application`.

3. W polu `Name` wpisz nazwę nowego katalogu, który będzie zawierał pliki projektu, w tym przypadku `Editor`. W polu `Create in` wybierz istniejący folder, w którym ten nowy katalog zostanie utworzony.

4. Wybierz moduły biblioteki Qt, które zostaną dołączone do projektu. W niniejszym przykładzie wystarczą dwa moduły zaznaczone domyślnie.

5. Z rozwijanej listy `Base class` wybierz klasę macierzystą klasy reprezentującej główne okno aplikacji, w tym przypadku `QWidget`. W polu `Class name` wpisz nazwę klasy reprezentującej główne okno aplikacji, w tym przypadku `Editor`. Odznacz pole `Create form` (tak, aby nie było zaznaczone).

6. Zatwierdź informacje zawarte w kolejnym okienku dialogowym.

7. Program QtCreator utworzył we wskazanym folderze katalog `Editor3`, w którym z kolei znalazły się pliki:

   - `editor.h` przeznaczony na deklarację klasy `Editor`;
   - `editor.cpp`, który będzie zawierał definicje metod tej klasy;
   - `main.cpp` zawierający funkcję `main`.

   Do każdego z tych plików QtCreator wpisał automatycznie szkielet jego zawartości.

8. Uzupełnij te pliki, aby wyglądały jak podano niżej.

9. Aby zbudować aplikację, kliknij okrągły przycisk młotka w lewym dolnym rogu.

10. W wyniku kompliacji i łączenia w katalogu `Editor` utworzony zostanie plik wykonywalny o takiej samej nazwie.

11. Aplikację można uruchomić z programu QtCreator klikając na okrągły przycisk zielonej strzałki lub zwyczajnie z systemu operacyjnego.

- Plik `editor.h`

```
#ifndef EDITOR_H
#define EDITOR_H

#include <QtGui>

class Editor: public QWidget {
  Q_OBJECT
public:
  Editor (QWidget *parent=0);
private:
  QTextEdit *textEdit;
  QPushButton *pushButton; };

#endif
```

- Plik `editor.cpp`

```
#include "editor.h"

Editor::Editor (QWidget *parent):
    QWidget (parent),
    textEdit (new QTextEdit ("Przykladowy tekst",this)),
    pushButton (new QPushButton ("Clear",this)) {
  connect (pushButton,SIGNAL (clicked ()),textEdit,SLOT (clear ())); }
```

- Plik `main.cpp`

```
#include "editor.h"

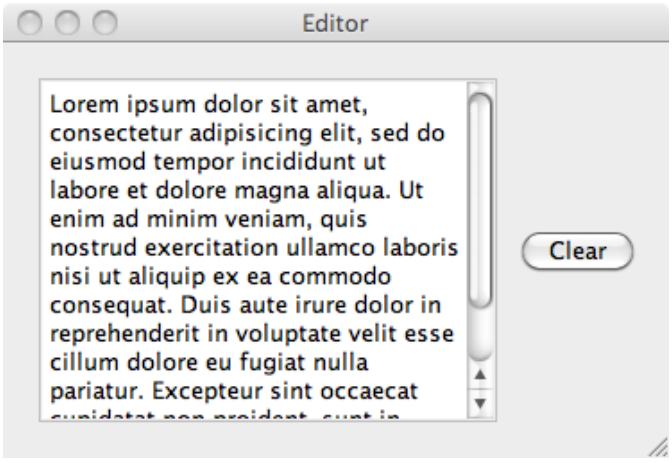
int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Editor editor;
  editor.show ();
  return application.exec (); }
```

W konstruktorze klasy `Editor`, przy wywołaniu konstruktorów obiektów `textEdit` i `pushButton` można na początku pominąć argument `this`. Spowoduje to utworzenie trzech okien - pustego okna głównego oraz oddzielnych okien z polem do edycji tekstu oraz przyciskiem. Argumenty `this` powodują, że pole edycyjne i przycisk znajdują się wewnątrz okna głównego. Jednak ich usytuowanie jest bardzo niewygodne. Aby je lepiej rozmieścić, w przykładzie `Editor4` wykorzystamy obiekty klasy `QLayout`.

6.4 Layouty

6.4.1 Editor4

Niniejszy przykład ilustruje wykorzystanie layoutów. Program wyświetla w jednym okienku pole do edycji tekstu oraz przycisk `Clear`, którego naciśnięcie usuwa cały tekst.



- Plik `editor.h`

```
#ifndef EDITOR_H
#define EDITOR_H
```

```
#include <QtGui>

class Editor: public QWidget {
  Q_OBJECT
public:
  Editor (QWidget *parent=0);
private:
  QTextEdit *textEdit;
  QPushButton *pushButton; };

#endif
```

- Plik `editor.cpp`

```
#include "editor.h"

Editor::Editor (QWidget *parent):
    QWidget (parent),
    textEdit (new QTextEdit ("Przykladowy tekst")),
    pushButton (new QPushButton ("Clear")) {
  connect (pushButton,SIGNAL (clicked ()),textEdit,SLOT (clear ()));
  QHBoxLayout *mainLayout=new QHBoxLayout;
  mainLayout->addWidget (textEdit);
  mainLayout->addWidget (pushButton);
  setLayout (mainLayout);
  setWindowTitle ("Editor"); }
```

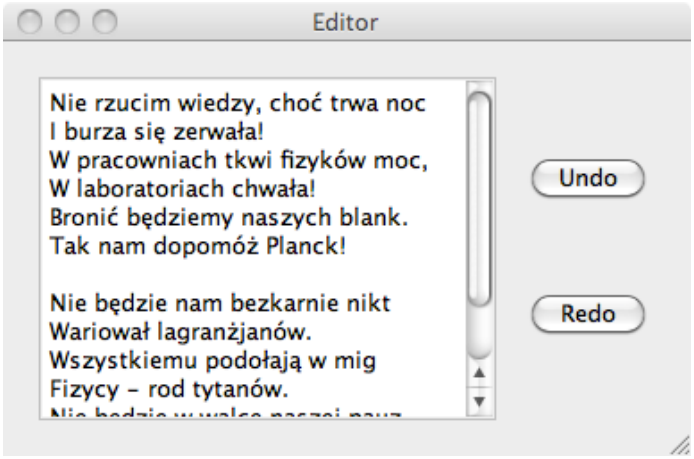
- Plik `main.cpp`

```
#include "editor.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Editor editor;
  editor.show ();
  return application.exec (); }
```

6.4.2 Editor5

Niniejszy przykład ilustruje zagnieżdżanie jednych layoutów w drugich. Program wyświetla w jednym okienku pole do edycji tekstu oraz przyciski `Undo` i `Redo`, służące do cofania i ponawiania edycji.



- Plik `editor.h`

```
#ifndef EDITOR_H
#define EDITOR_H

#include <QtGui>

class Editor: public QWidget {
  Q_OBJECT
public:
  Editor (QWidget *parent=0);
private:
  QTextEdit *textEdit;
  QPushButton *undoButton,*redoButton; };

#endif
```

- Plik `editor.cpp`

```
#include "editor.h"

Editor::Editor (QWidget *parent):
    QWidget (parent),
    textEdit (new QTextEdit ("Przykladowy tekst")),
    undoButton (new QPushButton ("Undo")),
    redoButton (new QPushButton ("Redo")) {
  connect (undoButton,SIGNAL (clicked ()),textEdit,SLOT (undo ()));
  connect (redoButton,SIGNAL (clicked ()),textEdit,SLOT (redo ()));
  QVBoxLayout *buttonsLayout=new QVBoxLayout;
  buttonsLayout->addWidget (undoButton);
  buttonsLayout->addWidget (redoButton);
  QHBoxLayout *mainLayout=new QHBoxLayout;
  mainLayout->addWidget (textEdit);
  mainLayout->addLayout (buttonsLayout);
  setLayout (mainLayout);
  setWindowTitle ("Editor"); }
```

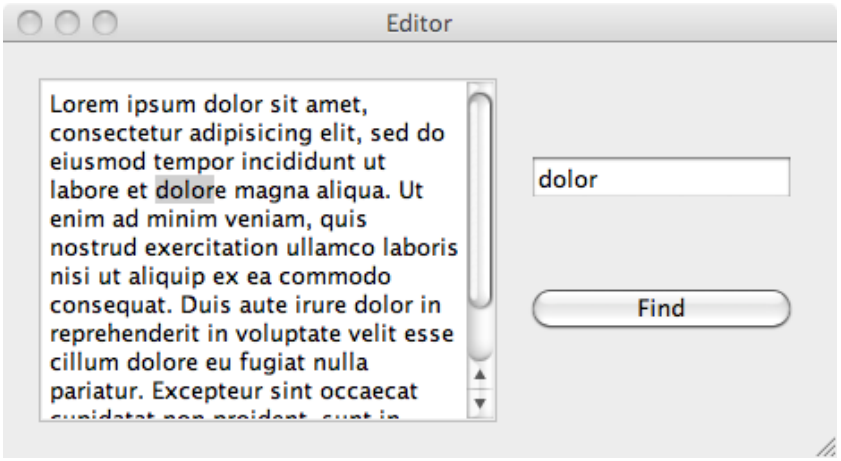
- Plik `main.cpp`

```
#include "editor.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Editor editor;
  editor.show ();
  return application.exec (); }
```

6.5 Własne gniazda

6.5.1 Editor6



- Plik `editor.h`

```
#ifndef EDITOR_H
#define EDITOR_H

#include <QtGui>

class Editor: public QWidget {
  Q_OBJECT
public:
  Editor (QWidget *parent=0);
private:
  QTextEdit *textEdit;
  QLineEdit *findEdit;
  QPushButton *findButton;
private slots:
  void find (); };

#endif
```

- Plik `editor.cpp`

```
#include "editor.h"
```

```
Editor::Editor (QWidget *parent):
    QWidget (parent),
    textEdit (new QTextEdit ("Przykladowy tekst")),
    findEdit (new QLineEdit),
    findButton (new QPushButton ("Find")) {
  connect (findButton,SIGNAL (clicked ()),this,SLOT (find ()));
  QVBoxLayout *buttonsLayout=new QVBoxLayout;
  buttonsLayout->addWidget (findEdit);
  buttonsLayout->addWidget (findButton);
  QHBoxLayout *mainLayout=new QHBoxLayout;
  mainLayout->addWidget (textEdit);
  mainLayout->addLayout (buttonsLayout);
  setLayout (mainLayout);
  setWindowTitle ("Editor"); }

void Editor::find () {
  if (!textEdit->find (findEdit->text ())) {
    textEdit->moveCursor (QTextCursor::Start);
    textEdit->find (findEdit->text ()); }}
```

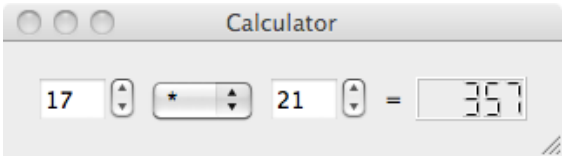
- Plik `main.cpp`

```
#include "editor.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Editor editor;
  editor.show ();
  return application.exec (); }
```

6.5.2 Calculator

Napisz program `calculator` pozwalający na dodawanie i mnożenie dwóch liczb całkowitych. Interfejs programu powinien wyglądać jak na rysunku.



Z rozwijanej listy można wybierać dodawanie lub mnożenie. Wynik powinien aktualizować się natychmiast po zmianie argumentów lub działania.

- Plik `calculator.h`

```
#ifndef CALCULATOR_H
#define CALCULATOR_H

#include <QtGui>
```

```
class Calculator: public QWidget {
  Q_OBJECT
public:
  Calculator (QWidget *parent=0);
private:
  QSpinBox *first,*second;
  QComboBox *operation;
  QLCDNumber *result;
private slots:
  void calculate (); };

#endif
```

- Plik `calculator.cpp`

```
#include "calculator.h"

Calculator::Calculator (QWidget *parent): QWidget (parent),
    first (new QSpinBox),second (new QSpinBox),
    operation (new QComboBox),result (new QLCDNumber) {
  operation->addItem ("+");
  operation->addItem ("*");
  connect (first,SIGNAL (valueChanged (int)),this,SLOT (calculate ()));
  connect (second,SIGNAL (valueChanged (int)),this,SLOT (calculate ()));
  connect (operation,SIGNAL (currentIndexChanged (int)),this,SLOT (calculate ()));
  QHBoxLayout *mainLayout=new QHBoxLayout;
  mainLayout->addWidget (first);
  mainLayout->addWidget (operation);
  mainLayout->addWidget (second);
  mainLayout->addWidget (new QLabel ("="));
  mainLayout->addWidget (result);
  setLayout (mainLayout);
  setWindowTitle ("Calculator"); }

void Calculator::calculate () {
  result->display (operation->currentIndex () ? first->value ()*second->value () : firs
```

- Plik `main.cpp`

```
#include "calculator.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Calculator calculator;
  calculator.show ();
  return application.exec (); }
```

6.5.3 Converter

Napisz program `converter` przeliczający złote na euro i odwrotnie. Program powinien wyświetlać dwa pola, jak na rysunku:

Po wpisaniu kwoty w dowolne pole w drugim polu powinna pojawiać się równowartość w drugiej walucie.

- Plik converter.h

```
#ifndef CONVERTER_H
#define CONVERTER_H

#include <QtGui>

class Converter: public QWidget {
  Q_OBJECT
public:
  Converter (QWidget *parent=0);
private:
  static const double rate;
  QDoubleSpinBox *plnSpin,*eurSpin;
private slots:
  void plntoeur ();
  void eurtopln (); };

#endif
```

- Plik converter.cpp

```
#include "converter.h"

const double Converter::rate=3.9175;

Converter::Converter (QWidget *parent):
    QWidget (parent),
    plnSpin (new QDoubleSpinBox),eurSpin (new QDoubleSpinBox) {
  connect (plnSpin,SIGNAL (valueChanged (double)),this,SLOT (plntoeur ()));
  connect (eurSpin,SIGNAL (valueChanged (double)),this,SLOT (eurtopln ()));
  QHBoxLayout *mainLayout=new QHBoxLayout;
  mainLayout->addWidget (new QLabel ("PLN:"));
  mainLayout->addWidget (plnSpin);
  mainLayout->addWidget (new QLabel ("EUR:"));
  mainLayout->addWidget (eurSpin);
  setLayout (mainLayout);
  setWindowTitle ("Converter"); }

void Converter::plntoeur () {
  eurSpin->setValue (plnSpin->value ()/rate); }
```

```
void Converter::eurtopln () {
  plnSpin->setValue (eurSpin->value ()*rate); }
```

- Plik `main.cpp`

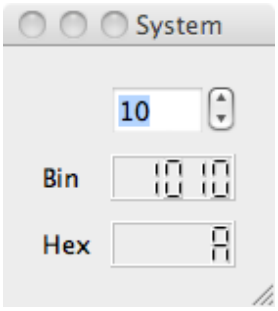
```
#include "converter.h"

int main (int argc, char *argv []) {
  QApplication application (argc,argv);
  Converter converter;
  converter.show ();
  return application.exec (); }
```

6.6 Sygnały z parametrami

6.6.1 System

Napisz program `system` przedstawiający zadaną dziesiętnie liczbę całkowitą nieujemną w formacie dwójkowym i szesnastkowym. Okienko programu powinno wyglądać jak na rysunku.



Wyniki w postaci binarnej i heksadecymalnej powinny aktualizować się natychmiast po zmianie liczby w postaci dziesiętnej.

- Plik `system.h`

```
#ifndef SYSTEM_H
#define SYSTEM_H

#include <QtGui>

class System: public QWidget {
  Q_OBJECT
public:
  System (QWidget *parent=0);
private:
  QSpinBox *dec;
  QLCDNumber *bin,*hex; };

#endif
```

- Plik `system.cpp`

```
#include "system.h"

System::System (QWidget *parent):
    QWidget (parent),dec (new QSpinBox),
    bin (new QLCDNumber),hex (new QLCDNumber) {
  bin->setMode (QLCDNumber::Bin);
  hex->setMode (QLCDNumber::Hex);
  connect (dec,SIGNAL (valueChanged (int)),bin,SLOT (display (int)));
  connect (dec,SIGNAL (valueChanged (int)),hex,SLOT (display (int)));
  QGridLayout *layout=new QGridLayout;
  layout->addWidget (dec,0,1);
  layout->addWidget (new QLabel ("Bin"),1,0);
  layout->addWidget (bin,1,1);
  layout->addWidget (new QLabel ("Hex"),2,0);
  layout->addWidget (hex,2,1);
  setLayout (layout);
  setWindowTitle ("System"); }
```

- Plik `main.cpp`

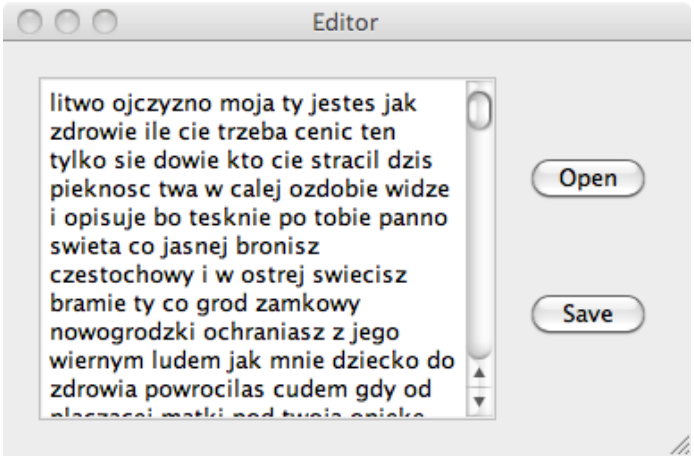
```
#include "system.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  System system;
  system.show ();
  return application.exec (); }
```

6.7 Pliki

6.7.1 Editor7

Niniejszy przykład ilustruje wykorzystanie łańcuchów, strumieni i plików, a także użycie okien do otwierania i zapisywania pliku. Program wyświetla w jednym okienku pole do edycji tekstu oraz przyciski `Open` i `Save` służące do otwierania i zapisywania plików.



- Plik `editor.h`

```
#ifndef EDITOR_H
#define EDITOR_H

#include <QtGui>

class Editor: public QWidget {
  Q_OBJECT
public:
  Editor (QWidget *parent=0);
private:
  QTextEdit *textEdit;
  QPushButton *openButton,*saveButton;
private slots:
  void open ();
  void save (); };

#endif
```

- Plik `editor.cpp`

```
#include "editor.h"

Editor::Editor (QWidget *parent):
    QWidget (parent),
    textEdit (new QTextEdit ("Przykladowy tekst")),
    openButton (new QPushButton ("Open")),
    saveButton (new QPushButton ("Save")) {
  connect (openButton,SIGNAL (clicked ()),this,SLOT (open ()));
  connect (saveButton,SIGNAL (clicked ()),this,SLOT (save ()));
  QVBoxLayout *buttonsLayout=new QVBoxLayout;
  buttonsLayout->addWidget (openButton);
  buttonsLayout->addWidget (saveButton);
  QHBoxLayout *mainLayout=new QHBoxLayout;
  mainLayout->addWidget (textEdit);
  mainLayout->addLayout (buttonsLayout);
  setLayout (mainLayout);
  setWindowTitle ("Editor"); }

void Editor::open () {
  QFile file (QFileDialog::getOpenFileName (this,"Open File",
                                            QDir::currentPath ()));
  file.open (QIODevice::ReadOnly | QIODevice::Text);
  textEdit->setPlainText (QTextStream (&file).readAll ());
  file.close (); }

void Editor::save () {
  QFile file (QFileDialog::getSaveFileName (this,"Save File",
                                            QDir::currentPath ()));
  file.open (QIODevice::WriteOnly | QIODevice::Text);
  QTextStream (&file)<<(textEdit->toPlainText ());
```

```
  file.close (); }
```

- Plik `main.cpp`

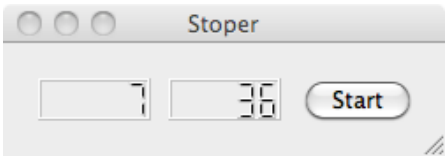
```
#include "editor.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Editor editor;
  editor.show ();
  return application.exec (); }
```

6.8 Czas

6.8.1 Stoper

Napisz program `stoper` będący najprostszą implementacją stopera odmierzającego sekundy i setne części sekund. Interfejs programu powinien wyglądać jak na rysunku.



Naciskanie przycisku powoduje na przemian uruchamianie i zatrzymywanie stopera. Za każdym uruchomieniem czas odmierzany jest na nowo od zera. Stoper zatrzymany wyświetla ostatnio zmierzony czas, a na jego przycisku widnieje napis `Start`. Stoper uruchomiony wyświetla szybko zmieniający się czas, a na jego przycisku znajduje się napis `Stop`.

- Plik `stoper.h`

```
#ifndef STOPER_H
#define STOPER_H

#include <QtGui>

class Stoper: public QWidget {
  Q_OBJECT
public:
  Stoper (QWidget *parent=0);
private:
  QLCDNumber *secondsLCD,*fractionsLCD;
  QPushButton *button;
  QTimer timer;
  int seconds,fractions;
private slots:
  void count ();
  void stoper (); };

#endif
```

- Plik `stoper.cpp`

```
#include <QtGui/QHBoxLayout>
#include "stoper.h"

Stoper::Stoper (QWidget *parent):
    QWidget (parent),
    secondsLCD (new QLCDNumber),fractionsLCD (new QLCDNumber),
    button (new QPushButton ("Start")),
    seconds (),fractions () {
  connect (&timer,SIGNAL (timeout ()),this,SLOT (count ()));
  connect (button,SIGNAL (clicked ()),this,SLOT (stoper ()));
  QHBoxLayout *mainLayout=new QHBoxLayout;
  mainLayout->addWidget (secondsLCD);
  mainLayout->addWidget (fractionsLCD);
  mainLayout->addWidget (button);
  setLayout (mainLayout);
  setWindowTitle ("Stoper"); }

void Stoper::count () {
  if (++fractions==100) {
    fractions=0;
    ++seconds; }
  secondsLCD->display (seconds);
  fractionsLCD->display (fractions); }

void Stoper::stoper () {
  if (timer.isActive ()) {
    timer.stop ();
    button->setText ("Start"); }
  else {
    button->setText ("Stop");
    seconds=fractions=0;
    timer.start (10); }}
```

- Plik `main.cpp`

```
#include "stoper.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Stoper stoper;
  stoper.show ();
  return application.exec (); }
```

6.9 Painter

6.9.1 Mandelbrot

Zbiór Mandelbrota jest to zbiór takich liczb zespolonych p, dla których ciąg określony relacją rekurencyjną

$$z_0 = 0 \qquad z_n = z_{n-1}^2 + p$$

nie dąży do nieskończoności. Można pokazać, że ciąg ten nie dąży do nieskończoności wtedy i tylko wtedy, gdy wszystkie jego wyrazy są co do modułu mniejsze od 2.

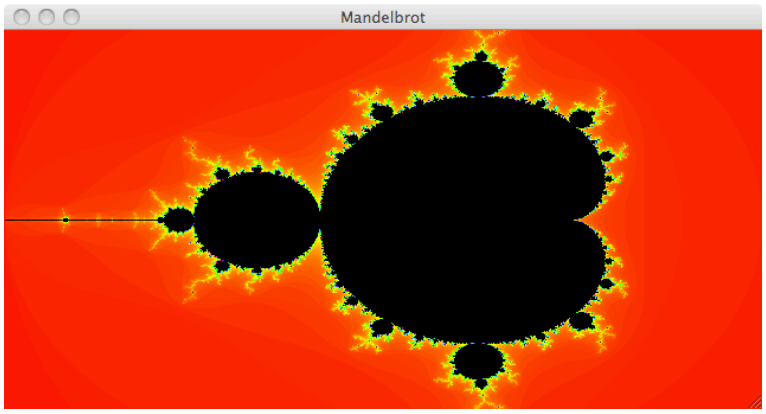
Chcąc wyrysować zbiór Mandelbrota przy pomocy komputera rozważamy siatkę punktów na płaszczyznie i dla każdego punktu siatki sprawdzamy, czy należy on do zbioru.

Dla uzyskania pożądanego efektu siatka powinna być dostatecznie gęsta. Jeżeli rysunek sporządzamy na ekranie, to najlepiej, aby węzły siatki odpowiadały kolejnym pikslom ekranu.

Oczywiście nie można sprawdzić dokładnie, czy dany punkt należy do zbioru Mandelbrota, ponieważ nie da się rozważyć nieskończenie wielu wyrazów ciągu. W praktyce wystarczy jednak wziąć skończoną liczbę N pierwszych wyrazów. Jeżeli wszystkie wyrazy o indeksach mniejszych od N są co do modułu mniejsze od 2, to uznajemy, że punkt należy do zbioru i zaznaczamy go na przykład kolorem czarnym.

Punkty nienależące do zbioru zwykle też zaznacza się kolorami, które oznaczają w tym przypadku stopień rozbieżności ciągu. Można na przykład uzależnić kolor od indeksu n wyrazu ciągu, który jako pierwszy stał się co do modułu większy od 2. Najwygodniej użyć do tego celu modelu kolorów HSV przyjmując kąt barwy równy $360° * n/N$.

Napisz program `mandelbrot` rysujący zbiór Mandelbrota. Program powinien być napisany tak, aby rysunek zbioru skalował się razem z okienkiem.



- Plik `mandelbrot.h`

```
#ifndef MANDELBROT_H
#define MANDELBROT_H

#include <QtGui>

class Mandelbrot: public QWidget {
  Q_OBJECT
public:
  Mandelbrot (QWidget *parent=0);
protected:
  void paintEvent (QPaintEvent *event); };
```

```
#endif
```

- Plik `mandelbrot.cpp`

```
#include "mandelbrot.h"
#include <complex>

Mandelbrot::Mandelbrot (QWidget *parent): QWidget (parent) {
  setWindowTitle ("Mandelbrot"); }

void Mandelbrot::paintEvent (QPaintEvent *event) {
  const int max=180;
  QPainter painter (this);
  for (int x=0;x<width ();x++)
    for (int y=0;y<height ();y++) {
      std::complex <double> p (3.*x/width ()-2,2.*y/height ()-1);
      std::complex <double> z=0;
      int n;
      for (n=0;n<max && norm (z)<4;n++)
        z=z*z+p;
      if (n<max) {
        QColor color;
        color.setHsv (360.*n/max,255,255);
        painter.setPen (color); }
      else
        painter.setPen (QColor ("black"));
      painter.drawPoint (x,y); }}
```

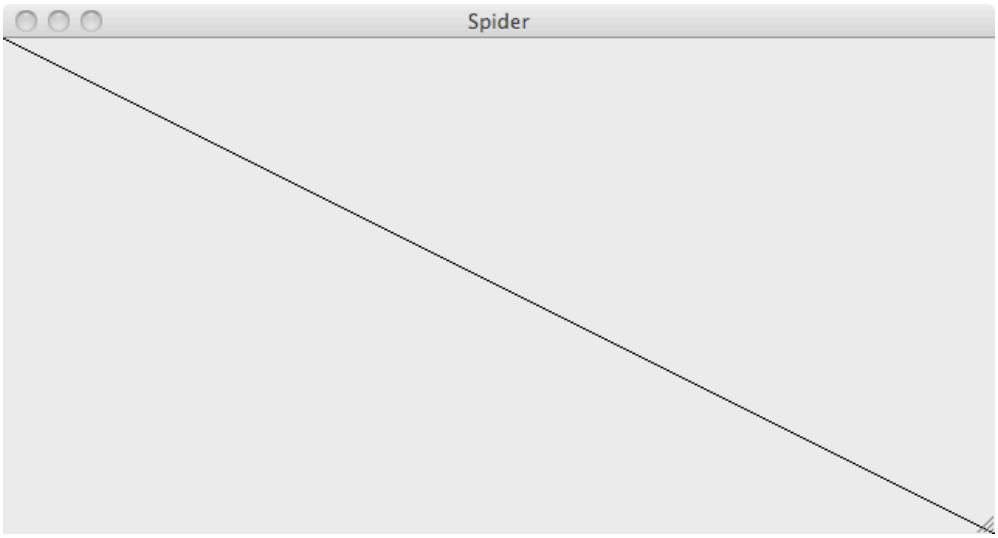
- Plik `main.cpp`

```
#include <QtGui>
#include "mandelbrot.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Mandelbrot mandelbrot;
  mandelbrot.show ();
  return application.exec (); }
```

6.9.2 Spider1

Napisz program `spider` wyświetlający okno z linią przebiegającą jak na rysunku.



Przy zmianie rozmiaru okna rysunek powinien być odświeżany tak, aby linia przebiegała zawsze od lewego górnego do prawego dolnego rogu.

- Plik `spider.h`

```
#ifndef SPIDER_H
#define SPIDER_H

#include <QtGui>

class Spider: public QWidget {
  Q_OBJECT
public:
  Spider (QWidget *parent=0);
protected:
  void paintEvent (QPaintEvent *event); };

#endif
```

- Plik `spider.cpp`

```
#include "spider.h"

Spider::Spider (QWidget *parent): QWidget (parent) {
  setWindowTitle ("Spider"); }

void Spider::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.drawLine (0,0,width (),height ()); }
```

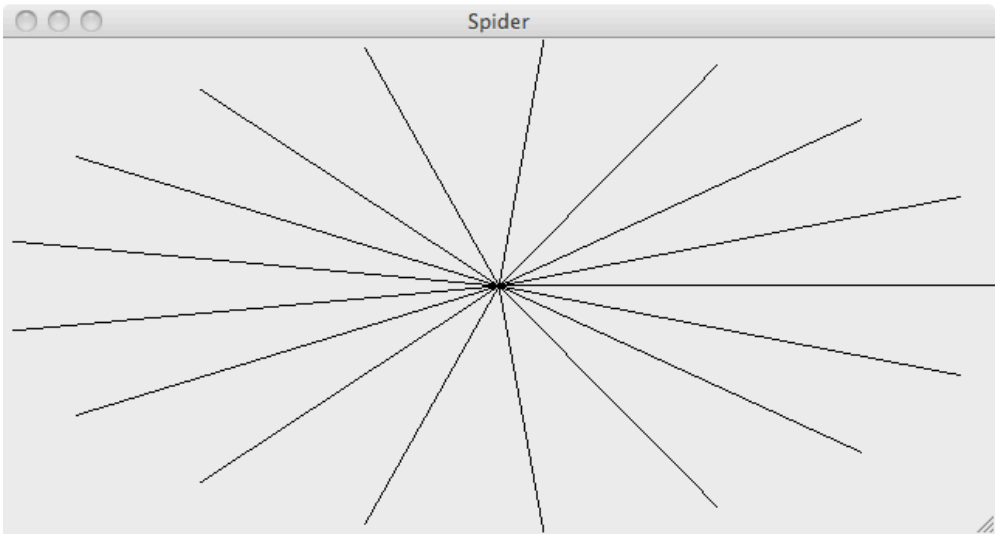
- Plik `main.cpp`

```
#include "spider.h"
```

```
int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Spider spider;
  spider.show ();
  return application.exec (); }
```

6.9.3 Spider2

Napisz program `spider` wyświetlający linie łączące środek siedemnastokąta foremnego z jego wierzchołkami. Rysunek powinien być wyśrodkowany w oknie i przeskalowany oddzielnie w poziomie i w pionie tak, aby zajmował całą szerokość i wysokość okna, jak przedstawiono poniżej na rysunku.



Przy zmianie rozmiarów okna rysunek powinien być odświeżany tak, aby zawsze wypełniał całe okno.

- Plik `spider.h`

```
#ifndef SPIDER_H
#define SPIDER_H

#include <QtGui>

class Spider: public QWidget {
  Q_OBJECT
public:
  Spider (QWidget *parent=0);
protected:
  void paintEvent (QPaintEvent *event); };

#endif
```

- Plik `spider.cpp`

```
#include <cmath>
#include "spider.h"

Spider::Spider (QWidget *parent): QWidget (parent) {
  setWindowTitle ("Spider"); }

void Spider::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.translate (width ()/2.,height ()/2.);
  painter.scale (width ()/200.,height ()/200.);
  int n=17;
  for (int i=0;i<n;++i)
    painter.drawLine (0,0,100*cos (2*M_PI*i/n),100*sin (2*M_PI*i/n)); }
```

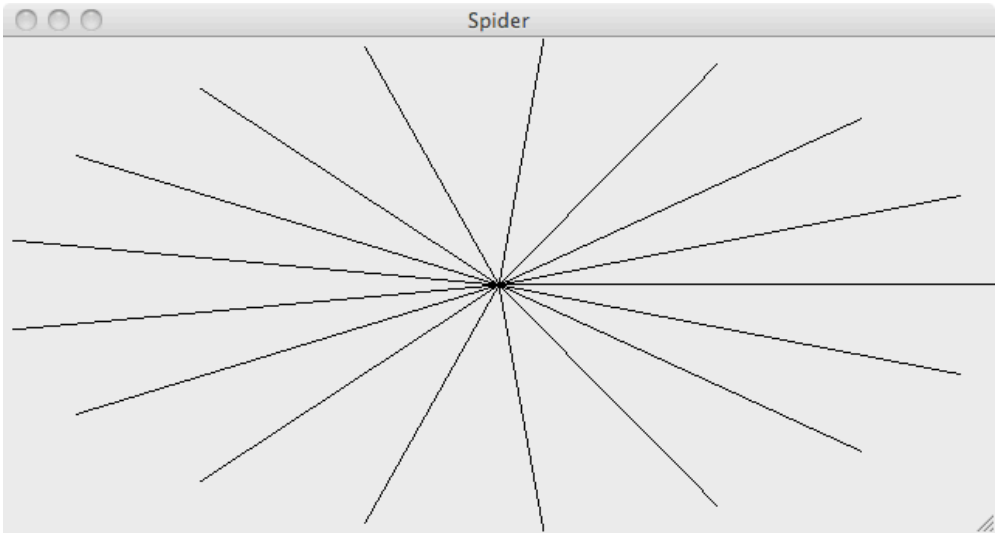
- Plik `main.cpp`

```
#include "spider.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Spider spider;
  spider.show ();
  return application.exec (); }
```

6.9.4 Spider3

Napisz program `spider` wyświetlający linie łączące środek siedemnastokąta foremnego z jego wierzchołkami. Wzór ten powinien się obracać ze stałą prędkością. Obrócony wzór powinien być wyśrodkowany w oknie i przeskalowany oddzielnie w poziomie i w pionie tak, aby zajmował całą szerokość i wysokość okna, jak przedstawiono poniżej:



Przy zmianie rozmiarów okna rysunek powinien być odświeżany tak, aby zawsze wypełniał całe okno.

- Plik `spider.h`

```
#ifndef SPIDER_H
#define SPIDER_H

#include <QtGui>

class Spider: public QWidget {
  Q_OBJECT
public:
  Spider (QWidget *parent=0);
protected:
  void paintEvent (QPaintEvent *event);
private:
  QTimer timer;
  double angle;
private slots:
  void rotate (); };

#endif
```

- Plik `spider.cpp`

```
#include <cmath>
#include "spider.h"

Spider::Spider (QWidget *parent): QWidget (parent),angle () {
  connect (&timer,SIGNAL (timeout ()),this,SLOT (rotate ()));
  timer.start (20); }

void Spider::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.translate (width ()/2.,height ()/2.);
  painter.scale (width ()/200.,height ()/200.);
  painter.rotate (angle);
  int n=17;
  for (int i=0;i<n;++i)
    painter.drawLine (0,0,100*cos (2*M_PI*i/n),100*sin (2*M_PI*i/n)); }

void Spider::rotate () {
  angle+=10*M_PI/180;
  update (); }
```

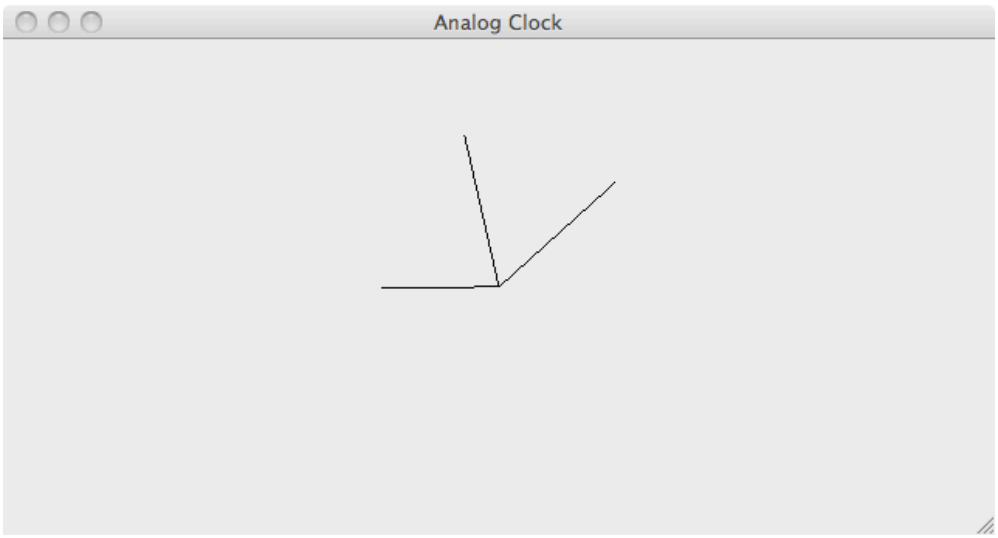
- Plik `main.cpp`

```
#include "spider.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Spider spider;
  spider.show ();
  return application.exec (); }
```

6.9.5 AnalogClock

Napisz program `AnalogClock` wyświetlający zegar z trzema wskazówkami i pokazujący rzeczywisty czas. Okno programu powinno wyglądać jak na rysunku.



Położenie wszystkich wskazówek powinno być aktualizowane co sekundę.

- Plik `analogclock.h`

```
#ifndef ANALOGCLOCK_H
#define ANALOGCLOCK_H

#include <QtGui>

class AnalogClock: public QWidget {
  Q_OBJECT
public:
  AnalogClock (QWidget *parent=0);
protected:
  void paintEvent (QPaintEvent *event); };

#endif
```

- Plik `analogclock.cpp`

```
#include <cmath>
#include "analogclock.h"

AnalogClock::AnalogClock (QWidget *parent): QWidget (parent) {
  QTimer *timer=new QTimer (this);
  connect (timer,SIGNAL (timeout()),this,SLOT (update ()));
  timer->start (1000);
  setWindowTitle ("Analog Clock"); }

void AnalogClock::paintEvent (QPaintEvent *event) {
```

```
QTime time=QTime::currentTime ();
QPainter painter (this);
painter.translate (width ()/2,height ()/2);
painter.scale (1,-1);
painter.save ();
painter.rotate (-360*(time.hour ()+(time.minute ()+time.second ()/60.)/60.)/12.);
painter.drawLine (0,0,0,75);
painter.restore ();
painter.save ();
painter.rotate (-360*(time.minute ()+time.second ()/60.)/60.);
painter.drawLine (0,0,0,100);
painter.restore ();
painter.save ();
painter.rotate (-360*time.second ()/60.);
painter.drawLine (0,0,0,100);
painter.restore (); }
```

- Plik `main.cpp`

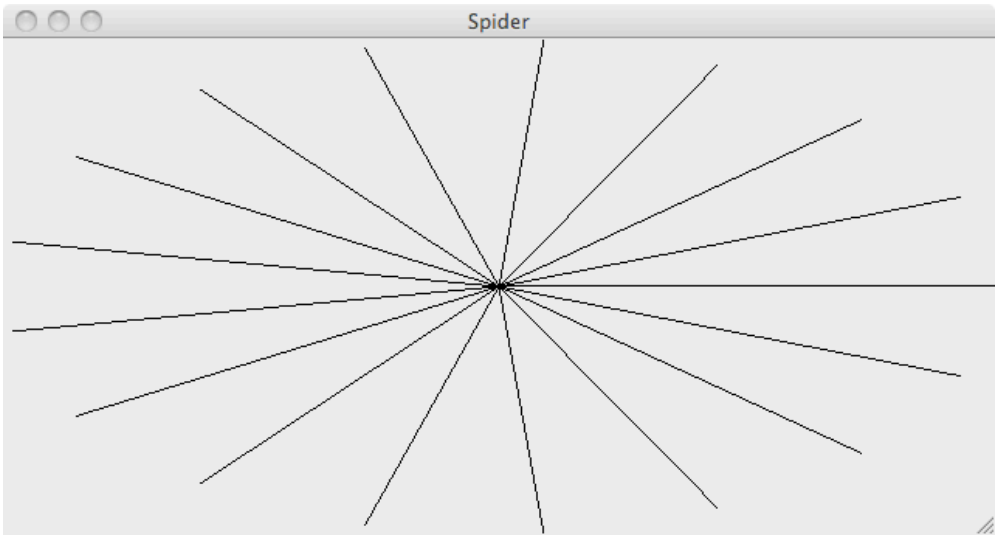
```
#include "analogclock.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  AnalogClock analogClock;
  analogClock.show ();
  return application.exec (); }
```

6.10 Zdarzenia

6.10.1 Spider4

Napisz program `spider` wyświetlający linie łączące środek siedemnastokąta foremnego z jego wierzchołkami. Wzór ten powinien się obracać z pewną prędkością. Obrócony wzór powinien być wyśrodkowany w oknie i przeskalowany oddzielnie w poziomie i w pionie tak, aby zajmował całą szerokość i wysokość okna, jak przedstawiono poniżej:



Przy zmianie rozmiarów okna rysunek powinien być odświeżany tak, aby zawsze wypełniał całe okno. Naciśnięcie klawisza `P` powinno zwiększać prędkość obrotu, zaś naciśnięcie klawisza `O` powinno ją zmniejszać.

- Plik `spider.h`

```
#ifndef SPIDER_H
#define SPIDER_H

#include <QtGui>

class Spider: public QWidget {
  Q_OBJECT
public:
  Spider (QWidget *parent=0);
protected:
  void keyPressEvent (QKeyEvent *event);
  void paintEvent (QPaintEvent *event);
private:
  QTimer timer;
  double angle,delta;
private slots:
  void rotate (); };

#endif
```

- Plik `spider.cpp`

```
#include <cmath>
#include "spider.h"

Spider::Spider (QWidget *parent): QWidget (parent),angle (),delta (10*M_PI/180) {
  connect (&timer,SIGNAL (timeout ()),this,SLOT (rotate ()));
  timer.start (20); }

void Spider::keyPressEvent (QKeyEvent *event) {
  if (event->key ()==Qt::Key_P)
    delta+=M_PI/180;
  if (event->key ()==Qt::Key_O)
    delta-=M_PI/180; }

void Spider::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.translate (width ()/2.,height ()/2.);
  painter.scale (width ()/200.,height ()/200.);
  painter.rotate (angle);
  int n=17;
  for (int i=0;i<n;++i)
    painter.drawLine (0,0,100*cos (2*M_PI*i/n),100*sin (2*M_PI*i/n)); }
```

```
void Spider::rotate () {
  angle+=delta;
  update (); }
```

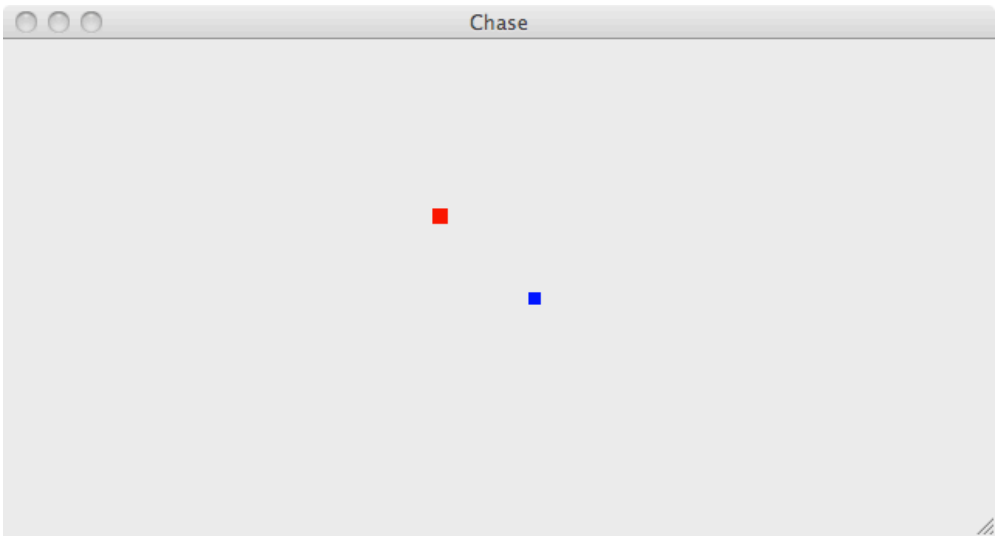
- Plik `main.cpp`

```
#include "spider.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Spider spider;
  spider.show ();
  return application.exec (); }
```

6.10.2 Chase1

Napisz program `chase` symulujący ucieczkę zająca przed niedźwiedziem. Program powinien wyświetlać mniejszy niebieski kwadrat symbolizujący zająca oraz większy czerwony kwadrat symbolizujący niedźwiedzia, jak na rysunku.



Zając porusza się z pewną prędkością w pewnym kierunku. W przypadku dotarcia do krawędzi okienka odbija się od niej zgodnie z prawem odbicia. Naciśnięcie klawiszy `O` lub `P` powoduje, że zając skręca nieznacznie w lewo lub prawo, przy zachowaniu długości wektora prędkości. Prędkość niedźwiedzia jest stała co do wartości i skierowana stale w stronę zająca.

- Plik `chase.h`

```
#ifndef CHASE_H
#define CHASE_H

#include <QtGui>

class Chase: public QWidget {
  Q_OBJECT
```

```
public:
  Chase (QWidget *parent=0);
protected:
  void keyPressEvent (QKeyEvent *event);
  void paintEvent (QPaintEvent *event);
private:
  QTimer timer;
  double rx,ry,vx,vy;
  double Rx,Ry,Vx,Vy;
private slots:
  void advance (); };

#endif
```

- Plik `chase.cpp`

```
#include <cmath>
#include "chase.h"

Chase::Chase (QWidget *parent):
    QWidget (parent),
    rx (),   ry (),vx (0.5),vy (),
    Rx (-10),Ry (),Vx (0.4),Vy () {
  connect (&timer,SIGNAL (timeout ()),this,SLOT (advance ()));
  timer.start (10);
  setWindowTitle ("Chase"); }

void Chase::keyPressEvent (QKeyEvent *event) {
  double phi=0;
  if (event->key ()==Qt::Key_P)
    phi=2*M_PI/180;
  if (event->key ()==Qt::Key_O)
    phi=-2*M_PI/180;
  double x=vx*cos (phi)-vy*sin (phi);
  double y=vx*sin (phi)+vy*cos (phi);
  vx=x; vy=y; }

void Chase::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.translate (width ()/2.,height ()/2.);
  painter.fillRect (rx-4,ry-4,8,8,Qt::blue);
  painter.fillRect (Rx-5,Ry-5,10,10,Qt::red); }

void Chase::advance () {
  rx+=vx; ry+=vy;
  if (rx<-width ()/2 || rx>width ()/2)
    vx*=-1;
  if (ry<-height ()/2 || ry>height ()/2)
    vy*=-1;
  double V=sqrt (Vx*Vx+Vy*Vy);
```

```
  double R=sqrt ((rx-Rx)*(rx-Rx)+(ry-Ry)*(ry-Ry));
  Vx=V*(rx-Rx)/R;
  Vy=V*(ry-Ry)/R;
  Rx+=Vx; Ry+=Vy;
  update (); }
```

- Plik `main.cpp`

```
#include "chase.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Chase chase;
  chase.show ();
  return application.exec (); }
```

6.10.3 Chase2

Napisz program `chase` symulujący ucieczkę zająca przed niedźwiedziem. Okno programu powinno wyglądać jak na rysunku.

Zając porusza się z pewną prędkością w pewnym kierunku. W przypadku dotarcia do krawędzi okienka odbija się od niej zgodnie z prawem odbicia. Naciśnięcie klawiszy `O` lub `P` powoduje, że zając skręca nieznacznie w lewo lub prawo, przy zachowaniu długości wektora prędkości. Prędkość niedźwiedzia jest stała co do wartości i skierowana stale w stronę zająca.

- Plik `chase.h`

```
#ifndef CHASE_H
#define CHASE_H

#include <QtGui>

class Chase: public QWidget {
```

```
  Q_OBJECT
public:
  Chase (QWidget *parent=0);
protected:
  void keyPressEvent (QKeyEvent *event);
  void paintEvent (QPaintEvent *event);
private:
  QImage rabbit,bear;
  QTimer timer;
  double rx,ry,vx,vy;
  double Rx,Ry,Vx,Vy;
private slots:
  void advance (); };

#endif
```

- Plik chase.cpp

```
#include <cmath>
#include "chase.h"

Chase::Chase (QWidget *parent):
    QWidget (parent),
    rabbit ("/Users/polbrat/Cwiczenia/Programowanie/Programy/Qt/Chase2/rabbit.png"),
    bear ("/Users/polbrat/Cwiczenia/Programowanie/Programy/Qt/Chase2/bear.png"),
    rx (),   ry (),vx (0.5),vy (),
    Rx (-10),Ry (),Vx (0.4),Vy () {
  connect (&timer,SIGNAL (timeout ()),this,SLOT (advance ()));
  timer.start (10);
  setWindowTitle ("Chase"); }

void Chase::keyPressEvent (QKeyEvent *event) {
  double phi=0;
  if (event->key ()==Qt::Key_P)
    phi=2*M_PI/180;
  if (event->key ()==Qt::Key_O)
    phi=-2*M_PI/180;
  double x=vx*cos (phi)-vy*sin (phi);
  double y=vx*sin (phi)+vy*cos (phi);
  vx=x; vy=y; }

void Chase::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.translate (width ()/2.,height ()/2.);
  painter.drawImage (rx-21,ry-27,rabbit);
  painter.drawImage (Rx-21,Ry-24,bear); }

void Chase::advance () {
  rx+=vx; ry+=vy;
  if (rx<-width ()/2 || rx>width ()/2)
```

```
    vx*=-1;
  if (ry<-height ()/2 || ry>height ()/2)
    vy*=-1;
  double V=sqrt (Vx*Vx+Vy*Vy);
  double R=sqrt ((rx-Rx)*(rx-Rx)+(ry-Ry)*(ry-Ry));
  Vx=V*(rx-Rx)/R;
  Vy=V*(ry-Ry)/R;
  Rx+=Vx; Ry+=Vy;
  update (); }
```

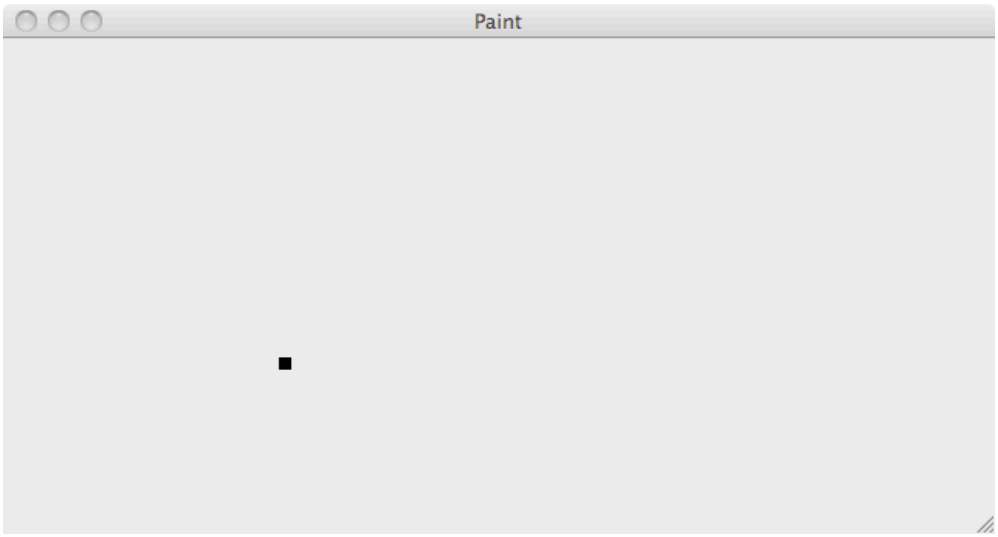
- Plik `main.cpp`

```
#include "chase.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Chase chase;
  chase.show ();
  return application.exec (); }
```

6.10.4 Paint1

Napisz program `paint`, który pod kursorem myszki wyświetla czarny kwadracik, jak na rysunku.



Kwadracik powinien przesuwać się razem z kursorem niezależnie od tego, czy któryś przycisk myszki jes naciśnięty, czy nie.

- Plik `paint.h`

```
#ifndef PAINT_H
#define PAINT_H

#include <QtGui>

class Paint: public QWidget {
```

```
  Q_OBJECT
public:
  Paint (QWidget *parent=0);
protected:
  void mouseMoveEvent (QMouseEvent *event);
  void paintEvent (QPaintEvent *event);
private:
  int x,y; };

#endif
```

- Plik `paint.cpp`

```
#include "paint.h"

Paint::Paint (QWidget *parent): QWidget (parent) {
  setMouseTracking (true);
  setWindowTitle ("Paint"); }

void Paint::mouseMoveEvent (QMouseEvent *event) {
  x=event->pos ().x ();
  y=event->pos ().y ();
  update (); }

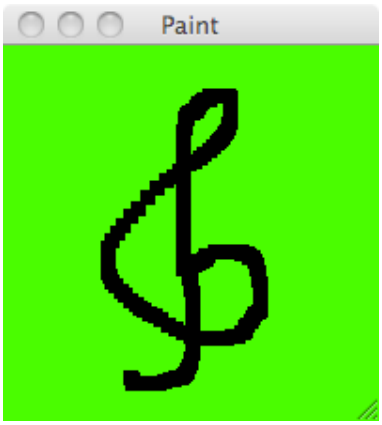
void Paint::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.fillRect (x-4,y-4,8,8,Qt::black); }
```

- Plik `main.cpp`

```
#include "paint.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Paint paint;
  paint.show ();
  return application.exec (); }
```

6.10.5 Paint2



- Plik `paint.h`

```
#ifndef PAINT_H
#define PAINT_H

#include <QtGui>

class Paint: public QWidget {
  Q_OBJECT
public:
  Paint (QWidget *parent=0);
protected:
  void mouseMoveEvent (QMouseEvent *event);
  void paintEvent (QPaintEvent *event);
private:
  QPixmap pixmap; };

#endif
```

- Plik `paint.cpp`

```
#include "paint.h"

Paint::Paint (QWidget *parent): QWidget (parent),pixmap (200,200) {
  pixmap.fill (Qt::green);
  resize (200,200);
  setWindowTitle ("Paint"); }

void Paint::mouseMoveEvent (QMouseEvent *event) {
  QPainter painter (&pixmap);
  painter.fillRect (event->pos ().x ()-4,event->pos ().y ()-4,8,8,Qt::black);
  update (); }

void Paint::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.drawPixmap (QPoint (0,0),pixmap); }
```

- Plik `main.cpp`

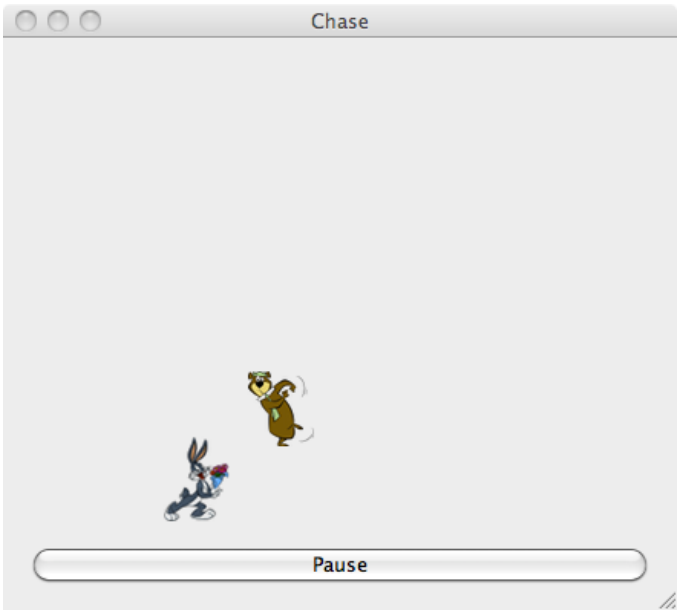
```
#include "paint.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Paint paint;
  paint.show ();
  return application.exec (); }
```

6.11 Własne widgety

6.11.1 Chase3

Napisz program `chase` symulujący ucieczkę zająca przed niedźwiedziem. Okno programu powinno wyglądać jak na rysunku.



Zając porusza się z pewną prędkością w pewnym kierunku. W przypadku dotarcia do krawędzi okienka odbija się od niej zgodnie z prawem odbicia. Naciśnięcie klawiszy `O` lub `P` powoduje, że zając skręca nieznacznie w lewo lub prawo, przy zachowaniu długości wektora prędkości. Prędkość niedźwiedzia jest stała co do wartości i skierowana stale w stronę zająca. Naciśnięcie przycisku `Pause` powoduje zatrzymanie zająca i niedźwiedzia aż do następnego naciśnięcia.

- Plik `chase.h`

```
#ifndef CHASE_H
#define CHASE_H

#include <QtGui>

class Chase: public QWidget {
  Q_OBJECT
public:
  Chase (QWidget *parent=0);
protected:
  void keyPressEvent (QKeyEvent *event);
  void paintEvent (QPaintEvent *event);
private:
  QImage rabbit,bear;
  QTimer timer;
  double rx,ry,vx,vy;
  double Rx,Ry,Vx,Vy;
public slots:
```

```
  void pause ();
private slots:
  void advance (); };

#endif
```

- Plik chase.cpp

```
#include <cmath>
#include "chase.h"

Chase::Chase (QWidget *parent):
    QWidget (parent),
    rabbit ("/Users/polbrat/Cwiczenia/Programowanie/Programy/Qt/Chase2/rabbit.png"),
    bear ("/Users/polbrat/Cwiczenia/Programowanie/Programy/Qt/Chase2/bear.png"),
    rx (),   ry (),vx (0.5),vy (),
    Rx (-10),Ry (),Vx (0.4),Vy () {
  connect (&timer,SIGNAL (timeout ()),this,SLOT (advance ()));
  timer.start (10);
  setFocusPolicy (Qt::StrongFocus);
  setWindowTitle ("Chase"); }

void Chase::keyPressEvent (QKeyEvent *event) {
  double phi=0;
  if (event->key ()==Qt::Key_P)
    phi=2*M_PI/180;
  if (event->key ()==Qt::Key_O)
    phi=-2*M_PI/180;
  double x=vx*cos (phi)-vy*sin (phi);
  double y=vx*sin (phi)+vy*cos (phi);
  vx=x; vy=y; }

void Chase::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.translate (width ()/2.,height ()/2.);
  painter.drawImage (rx-21,ry-27,rabbit);
  painter.drawImage (Rx-21,Ry-24,bear); }

void Chase::pause () {
  if (timer.isActive ())
    timer.stop ();
  else
    timer.start (); }

void Chase::advance () {
  rx+=vx; ry+=vy;
  if (rx<-width ()/2 || rx>width ()/2)
    vx*=-1;
  if (ry<-height ()/2 || ry>height ()/2)
    vy*=-1;
```

```
  double V=sqrt (Vx*Vx+Vy*Vy);
  double R=sqrt ((rx-Rx)*(rx-Rx)+(ry-Ry)*(ry-Ry));
  Vx=V*(rx-Rx)/R;
  Vy=V*(ry-Ry)/R;
  Rx+=Vx; Ry+=Vy;
  update (); }
```

- Plik game.h

```
#ifndef GAME_H
#define GAME_H

#include <QtGui>

class Game: public QWidget {
  Q_OBJECT
public:
  Game (QWidget *parent=0); };

#endif
```

- Plik game.cpp

```
#include "chase.h"
#include "game.h"

Game::Game (QWidget *parent): QWidget (parent) {
  Chase *chase=new Chase;
  QPushButton *pause=new QPushButton ("Pause");
  connect (pause,SIGNAL (clicked ()),chase,SLOT (pause ()));
  chase->setMinimumSize (400,300);
  QVBoxLayout *mainLayout=new QVBoxLayout;
  mainLayout->addWidget (chase);
  mainLayout->addWidget (pause);
  setLayout (mainLayout);
  setWindowTitle ("Chase"); }
```

- Plik main.cpp

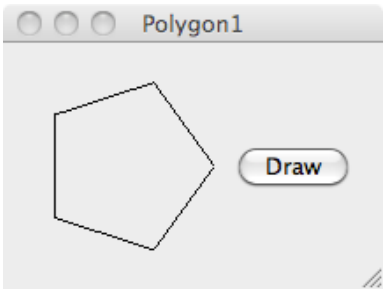
```
#include "game.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Game game;
  game.show ();
  return application.exec (); }
```

6.11.2 Polygon1

Niniejszy przykład ilustruje umieszczanie w okienku samodzielnie zdefiniowanych widgetów.

Program wyświetla w jednym okienku przycisk oraz samodzielnie zdefiniowane pole do rysowania. Przy każdym odświeżeniu pola jest w nim rysowany wielokąt foremny o losowej liczbie wierzchołków. Pole jest odświeżane po naciśnięciu przycisku oraz zmianie rozmiarów okienka itd.



- Plik `canvas.h`

```
#ifndef CANVAS_H
#define CANVAS_H

#include <QtGui>

class Canvas: public QWidget {
  Q_OBJECT
public:
  Canvas (QWidget *parent=0);
protected:
  void paintEvent (QPaintEvent *event); };

#endif
```

- Plik `canvas.cpp`

```
#include <cstdlib>
#include "canvas.h"

Canvas::Canvas (QWidget *parent): QWidget (parent) {}

void Canvas::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.translate (width ()/2.,height ()/2.);
  painter.scale (width ()/200.,height ()/200.);
  int n=3+random ()%7;
  QPointF previous,next;
  for (int i=0;i<=n;i++) {
    next=100*QPointF (cos (2*M_PI*i/n),sin (2*M_PI*i/n));
    if (i)
      painter.drawLine (previous,next);
    previous=next; }}
```

- Plik `polygon1.h`

```
#ifndef POLYGON1_H
#define POLYGON1_H

#include <QtGui>
#include "canvas.h"

class Polygon1: public QWidget {
  Q_OBJECT
public:
  Polygon1 (QWidget *parent=0);
private:
  Canvas *canvas;
  QPushButton *pushButton; };

#endif
```

- Plik `polygon1.cpp`

```
#include "polygon1.h"

Polygon1::Polygon1 (QWidget *parent):
    QWidget (parent),canvas (new Canvas),pushButton (new QPushButton ("Draw")) {
  canvas->setMinimumSize (QSize (100,100));
  QHBoxLayout *mainLayout=new QHBoxLayout;
  mainLayout->addWidget (canvas);
  mainLayout->addWidget (pushButton);
  setLayout (mainLayout);
  connect (pushButton,SIGNAL (clicked ()),canvas,SLOT (update ()));
  setWindowTitle ("Polygon1"); }
```

- Plik `main.cpp`

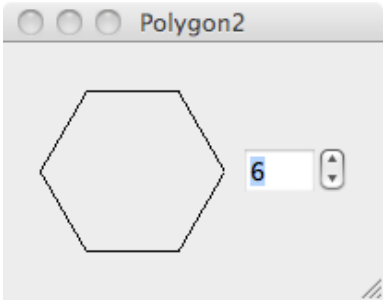
```
#include <QtGui>
#include "polygon1.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Polygon1 polygon1;
  polygon1.show ();
  return application.exec (); }
```

6.11.3 Polygon2

Niniejszy przykład ilustruje jeden ze sposobów przekazywania parametrów pomiędzy widgetami.

Program wyświetla w jednym okienku spinboks oraz samodzielnie zdefiniowane pole do rysowania. Wpisanie w spinboksie liczby powoduje wyślwietlenie w polu do rysowania wielokąta foremnego o wpisanej liczbie wierzchołków.



- Plik canvas.h

```
#ifndef CANVAS_H
#define CANVAS_H

#include <QtGui>

class Canvas: public QWidget {
  Q_OBJECT
public:
  Canvas (QWidget *parent=0);
protected:
  void paintEvent (QPaintEvent *event);
private:
  int number;
public slots:
  void setNumber (int number); };

#endif
```

- Plik canvas.cpp

```
#include "canvas.h"

Canvas::Canvas (QWidget *parent): QWidget (parent),number (7) {}

void Canvas::setNumber (int number) {
  Canvas::number=number;
  update (); }

void Canvas::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.translate (width ()/2.,height ()/2.);
  painter.scale (width ()/200.,height ()/200.);
  QPointF previous,next;
  for (int i=0;i<=number;i++) {
    next=100*QPointF (cos (2*M_PI*i/number),sin (2*M_PI*i/number));
    if (i)
      painter.drawLine (previous,next);
    previous=next; }}
```

- Plik `polygon2.h`

```
#ifndef POLYGON2_H
#define POLYGON2_H

#include <QtGui>
#include "canvas.h"

class Polygon2: public QWidget {
  Q_OBJECT
public:
  Polygon2 (QWidget *parent=0);
private:
  Canvas *canvas;
  QSpinBox *spinBox; };

#endif
```

- Plik `polygon2.cpp`

```
#include "polygon2.h"

Polygon2::Polygon2 (QWidget *parent):
    QWidget (parent),canvas (new Canvas),spinBox (new QSpinBox) {
  spinBox->setRange (3,9);
  canvas->setNumber (spinBox->value ());
  canvas->setMinimumSize (QSize (100,100));
  QHBoxLayout *mainLayout=new QHBoxLayout;
  mainLayout->addWidget (canvas);
  mainLayout->addWidget (spinBox);
  setLayout (mainLayout);
  connect (spinBox,SIGNAL (valueChanged (int)),canvas,SLOT (setNumber (int)));
  setWindowTitle ("Polygon2"); }
```

- Plik `main.cpp`

```
#include <QtGui>
#include "polygon2.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Polygon2 polygon2;
  polygon2.show ();
  return application.exec (); }
```

6.12 QtDesigner

6.12.1 Editor8

- Plik `editor.h`

```
#ifndef EDITOR_H
#define EDITOR_H

#include "ui_editor.h"

class Editor: public QWidget,private Ui::Editor {
  Q_OBJECT
public:
  Editor (QWidget *parent=0); };

#endif
```

- Plik `editor.cpp`

```
#include "editor.h"

Editor::Editor (QWidget *parent): QWidget (parent) {
  setupUi (this);
  textEdit->setPlainText ("Poczatkowy tekst"); }
```

- Plik `main.cpp`

```
#include <QtGui>
#include "editor.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Editor editor;
  editor.show ();
  return application.exec (); }
```

6.12.2 Polygon3

- Plik `canvas.h`

```
#ifndef CANVAS_H
#define CANVAS_H

#include <QtGui>

class Canvas: public QWidget {
  Q_OBJECT
public:
  Canvas (QWidget *parent=0);
protected:
  void paintEvent (QPaintEvent *event);
private:
  int number;
public slots:
  void setNumber (int number); };
```

```
#endif
```

- Plik canvas.cpp

```
#include "canvas.h"

Canvas::Canvas (QWidget *parent): QWidget (parent),number (7) {}

void Canvas::setNumber (int number) {
  Canvas::number=number;
  update (); }

void Canvas::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.translate (width ()/2.,height ()/2.);
  painter.scale (width ()/200.,height ()/200.);
  QPointF previous,next;
  for (int i=0;i<=number;i++) {
    next=100*QPointF (cos (2*M_PI*i/number),sin (2*M_PI*i/number));
    if (i)
      painter.drawLine (previous,next);
    previous=next; }}
```

- Plik polygon.h

```
#ifndef POLYGON_H
#define POLYGON_H

#include "ui_polygon.h"

class Polygon: public QWidget,private Ui::Polygon {
  Q_OBJECT
public:
  Polygon (QWidget *parent=0); };

#endif
```

- Plik polygon.cpp

```
#include "polygon.h"

Polygon::Polygon (QWidget *parent): QWidget (parent) {
  setupUi (this);
  canvas->setNumber (spinBox->value ()); }
```

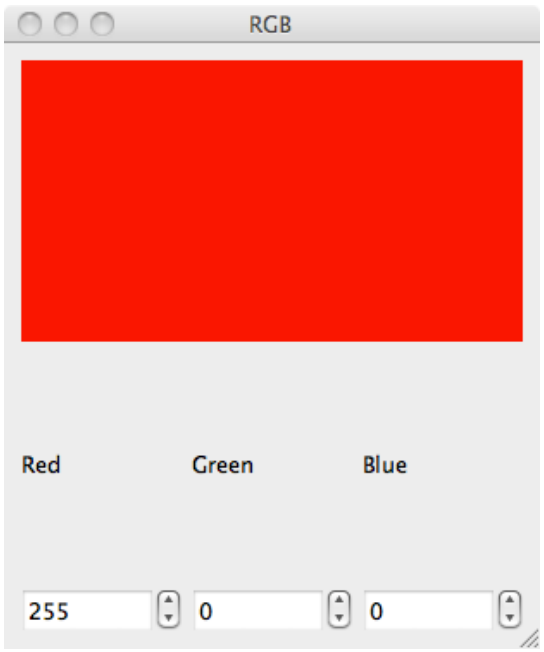
- Plik main.cpp

```
#include <QtGui>
#include "polygon.h"
```

```
int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  Polygon polygon;
  polygon.show ();
  return application.exec (); }
```

6.12.3 RGB

Napisz program `rgb` wyświetlający zadany kolor RGB. Interfejs graficzny powinien wyglądać jak na rysunku.



Wyświetlany kolor powinien być odświeżany po każdej zmianie wartości w polach do edycji.

- Plik `canvas.h`

```
#ifndef CANVAS_H
#define CANVAS_H

#include <QtGui>

class Canvas: public QWidget {
  Q_OBJECT
public:
  Canvas (QWidget *parent=0);
  void setColor (const QColor &color);
protected:
  void paintEvent (QPaintEvent *event);
private:
  QColor color; };

#endif
```

- Plik canvas.cpp

```
#include "canvas.h"

Canvas::Canvas (QWidget *parent): QWidget (parent),color () {}

void Canvas::setColor (const QColor &color) {
  this->color=color;
  update (); }

void Canvas::paintEvent (QPaintEvent *event) {
  QPainter painter (this);
  painter.fillRect (0,0,width (),height (),color); }
```

- Plik rgb.h

```
#ifndef RGB_H
#define RGB_H

#include "ui_rgb.h"

class RGB: public QWidget,private Ui::RGB {
  Q_OBJECT
public:
  RGB (QWidget *parent=0);
public slots:
  void changeColor (); };

#endif
```

- Plik rgb.cpp

```
#include "rgb.h"

RGB::RGB (QWidget *parent): QWidget (parent) {
  setupUi (this); }

void RGB::changeColor () {
  canvas->setColor (QColor (redSpinBox->value (),
                            greenSpinBox->value (),
                            blueSpinBox->value ())); }
```

- Plik main.cpp

```
#include <QtGui>
#include "rgb.h"

int main (int argc,char *argv []) {
  QApplication application (argc,argv);
  RGB rgb;
  rgb.show ();
  return application.exec (); }
```