Rekurencja

Rekurencja zwana również rekursją, polega na wywołaniu przez funkcję samej siebie. Algorytmy rekurencyjne zastępują często iteracyjne (za pomocą pętli) gdyż rozwiązanie niektórych problemów za pomocą algorytmu rekurencyjnego jest znacznie wygodniejsze. Niekiedy problemy rozwiązywane za pomocą rekurencji będą miały nieznacznie wolniejszy czas od iteracyjnego odpowiednika (wiąże się to z wywoływaniem funkcji). Aby zilustrować działanie algorytmu rekurencyjnego prześledźmy program wyznaczający sumę *n* kolejnych liczb naturalnych. W programie napiszemy funkcję rekurencyjną `suma(n)`, która zwróci sumę *n* kolejnych liczb naturalnych. Zauważmy, że suma *n* kolejnych liczb naturalnych może być przedstawiona w następujący sposób: $\sum_{i=1}^{n} i = n + \sum_{i=1}^{n-1} i$. Zatem, w programie, zależność ta odpowiada rekurencyjnemu wywołaniu funkcji `suma`: `suma( n ) = n + suma( n - 1 )`. Pisząc funkcję rekurencyjną należy pamiętać o tym, że taka funkcja musi zawierać warunek zakończenia rekurencji po spełnieniu którego nie następuje już kolejne wywołanie tej samej funkcji ale zwrócenie wyniku dla przypadku granicznego. Zatem implementacja naszej funkcji `suma( n )` będzie mieć następującą postać:

```
unsigned long suma( unsigned long n )
{
if ( n == 1 ) return 1;  // warunek zakończenia rekurencji dla n równego 1

return n + suma( n - 1 ); // rekurencyjne wywołanie funkcji suma
}
```

Funkcje operatorowe

Klasy definiowane przez użytkownika muszą być co najmniej tak samo dobrymi typami jak typy wbudowane (np. `int`). Oznacza to że, muszą dać się efektywnie zaimplementować oraz muszą dać się wygodnie używać. To drugie wymaga, by twórca klasy mógł zdefiniować tak zwane funkcje operatorowe czyli specjalne funkcje wywoływane poprzez stosowanie określonych operatorów. Podstawowym celem stosowania funkcji operatorowych w klasach jest stworzenie mechanizmu wykonywania operacji na obiektach w taki sam sposób jak na typach wbudowanych (np. dodawanie etc.). Aby to osiągnąć, język C++ umożliwia tak zwane przeciążanie operatorów czyli definiowanie ich w sposób odpowiedni dla własnych klas. Przeciążone operatory są implementowane jako metody składowe klasy lub standardowe funkcje na zewnątrz danej klasy. Nazwa funkcji operatorowej składa się ze słowa `operator` i nazwy operatora (np. `operator=`). Funkcja operatorowa musi mieć co najmniej jeden argument będący klasą bądź referencją do klasy. Operatory, które nie modyfikują stanu obiektu i nie potrzebują dostępu do prywatnych składowych klasy tworzymy jako funkcje będące na zewnątrz klasy. Implementację operatorów prześledzimy na przykładach. Załóżmy, że mamy klasę zdefiniowaną następująco:

```
class ValueContainer
{
 int m_value;
 public:
  ValueContainer(int value): m_value(value){}
  int get_value() const {return m_value;}
}
```

Operator przypisania (metoda składowa klasy)

```
ValueContainer& operator= (const ValueContainer& rhs)
{
 if ( this != &rhs )
 {
   m_value = rhs.m_value;
 }
return *this;}
```

Operator dodawania (funkcja zewnętrzna)

```
ValueContainer operator+ (const ValueContainer& lhs, const ValueContainer& rhs)
{
 return ValueContainer(lhs.get_value() + rhs.get_value());
}
```

Operator pre-inkrementacji (metoda składowa klasy)

```
ValueContainer& operator++ () //pre-inkrementacja
{
 ++m_value;
 return *this;
}
```

Operator post-inkrementacji (metoda składowa klasy)

```
ValueContainer& operator++ (int) //post-inkrementacja
{
  ValueContainer temp = *this;
  ++*this;
  return temp;
}
```

Operatory wejścia i wyjścia (funkcje zewnętrzne)

```
istream& operator>> ( istream& in, ValueContainer& rhs )
{// operator wejścia
 int v;
 in >> v;
 rhs.set_v(v);
 return in;
}

ostream& operator<< ( ostream& out, const ValueContainer& rhs )
{// operator wyjścia
 out << rhs.get_v();
 return out;
}
```

Operator porównania (funkcja zewnętrzna)

```
bool operator== (const ValueContainer& lhs, const ValueContainer& rhs )
{
 return lhs.get_value() == rhs.get_value();
}
```

Dla przykładu, mając zdefiniowane operatory +, =, << dla klasy `ValueContainer` możemy wykonać dodawanie dwóch obiektów tej klasy oraz wypisanie wyniku na ekranie w następujący sposób:

```
ValueContainer a(10);
ValueContainer b(5);
ValueContainer c = a + b;
cout << c;
```

zadania

1. Napisz program, który wczyta z klawiatury liczbę całkowitą a następnie za pomocą funkcji rekurencyjnej wyliczy wartość silni tej liczby.

2. Napisz definicję klasy, która reprezentuje liczbę zespoloną. Klasa powinna posiadać: dwa pola reprezentujące odpowiednio część rzeczywistą i urojoną liczby zespolonej. Konstruktor z dwoma argumentami i przypisanymi im wartościami domyślnymi. Gettery i settery dla części rzeczywistej i urojonej. Zaimplementuj następujące operatory: dodawanie, inkrementacja, wejście i wyjście, przypisanie i porównanie. W funkcji `main()` utwórz dwa obiekty reprezentujące liczby zespolone i wykonaj na nich

działania za pomocą zdefiniowanych funkcji operatorowych. Wypisz wynik na ekranie posługując się zdefiniowanym operatorem wyjścia.

3. Napisz program, który posługując się funkcją rekurencyjną wypisze w odwrotnej kolejności wprowadzony z klawiatury napis.

4. Metoda bisekcji zwana również metodą równego podziału jest jedną z numerycznych metod rozwiązywania równań nieliniowych. Jej działanie opiera się na twierdzeniu Darboux, które głosi, że jeżeli funkcja ciągła $f(x)$ ma na końcach przedziału domkniętego wartości różnych znaków, to wewnątrz tego przedziału, istnieje co najmniej jeden pierwiastek równania $f(x)=0$. Aby można było zastosować metodę równego podziału, funkcja $f(x)$ musi spełniać następujące założenia: po pierwsze funkcja $f(x)$ jest ciągła w przedziale domkniętym [a; b], po drugie funkcja przyjmuje różne znaki na końcach przedziału czyli $f(a)f(b)<0$. Zatem zgodnie z metodą bisekcji algorytm wyznaczania pierwiastka funkcji $f(x)$ w przedziale [a; b] przebiega następująco:
    a. Sprawdzenie, czy pierwiastkiem równania jest punkt $x_1 = \frac{a+b}{2}$ czyli czy $f(x_1)=0$. Jeżeli tak jest to algorytm kończy działanie, a punkt x_1 jest szukanym miejscem zerowym.
    b. W przeciwnym razie, dopóki nie osiągniemy żądanej dokładności, czyli dopóki $|a - b| > \varepsilon$ powtarzamy:
        i. Zgodnie ze wzorem z punktu (a) ponownie wyznaczane jest x_1, dzieląc przedział [a; b] na dwa mniejsze przedziały: [a; x_1] oraz [x_1; b].
        ii. Z pośród dwóch mniejszych przedziałów wybieramy ten którego końce mają różne znaki, czyli jeśli $f(a)f(x_1)<0$ to wybieramy przedział [a; x_1], zaś w przypadku gdy $f(x_1)f(b)<0$ wybieramy przedział [x_1; b].
    c. Po osiągnięciu żądanej dokładności algorytm kończy działanie, a szukany pierwiastek równania wynosi $x_1=(a+b)/2$.

    Napisz program, który implementuje za pomocą funkcji rekurencyjnej algorytm bisekcji a nastepnie wyznaczy pierwiastek równania $3x^2 + 7x + 1 = 0$ w przedziale [-1.0; 1.0].
5. Napisz program, który rekurencyjnie sprawdza czy podana liczba naturalna jest pierwsza.
6. Napisz program, który za pomocą rekurencyjnej funkcji wypisze na ekranie wszystkie permutacje elementów wektora.