

## Programowanie Obiektowe

Autor: Hubert Zając, Elektronika 283056

<https://github.com/huuubertz/STL.git>

### Zadanie 1

#### OPIS:

Wykorzystane funkcje dla:

Vector: size(), resize(), (constructor), back(), front(), push\_back(), pop\_back(), clear()

List: size(), push\_front(), push\_back(), begin(), end(), clear(), iterator()

Do kodu podajemy liczby dodatnie. Gdy podamy ujemną liczbę program kończy swoje działanie i pokazuje nam, jaka liczba spowodowała koniec programu. Działanie kodu jest opisane w komentarzach w poniższym kodzie.

#### KOD:

```
void sample_stl_program(){  
  
    // definiujemy liste i vector  
    std::list<int> lista;  
    std::vector<float> wektor(10);  
  
    // przykładowe dane przekazywane do listy  
    int liczba;  
    float suma=0;  
  
    // przekazujemy dane dopoki liczba nie będzie mniejsza od 0  
    std::cout << "Podaj liczby do przefiltrowania" << std::endl;  
    do{  
        std::cin >> liczba;  
  
        // zapisuj każdą podaną liczbę do listy  
        if (liczba % 2 == 0){  
            lista.push_front(liczba);  
        }  
        else lista.push_back(liczba);  
  
        // Jeżeli lista > 10 to zrob sumę z tych liczb i przekaz ją do wektora  
        if (lista.size() >= 10){  
            std::cout << lista.size() << " danych zostało wpisanych" <<  
std::endl;  
  
            // sumujemy te 10 liczb  
            // ustawiamy iterator listy na jej początek  
            std::list<int>::iterator iterator_listy = lista.begin();  
            while (iterator_listy != lista.end()) {  
  
                //std::cout << *iterator_listy << std::endl;  
                // sumujemy 10 liczb z listy  
                suma += *iterator_listy;  
                ++iterator_listy;  
            }  
        }  
    } while (liczba > 0);  
}
```

```

        // wyczyść listę
        lista.clear();

        // zapisujemy do wektora sumę 10 liczb z listy i dzielimy przez 10
        wektor.push_back(suma / 10);
        std::cout << wektor.at(1) << std::endl;

        // Jeżeli długość wektora będzie równa 10 to zwiększamy jego
rozmiar do 20;
        if (wektor.size() == 10){
            wektor.resize(20);
        }

        // wypisujemy liczbę na wyjście
        std::cout << wektor.front() << std::endl;

        // gdy wektor ma wielkość 20 ściągamy z niego dane i sumujemy
        if (wektor.size() == 20){
            int idx = 0;
            suma = 0;
            while (idx != wektor.size()){
                suma += wektor.back();
                wektor.pop_back();
            }
            //std::cout << suma << std::endl;

            // dodaj do wektora element suma podzielony przez 20
            wektor.push_back(suma / 20);

            // wyświetl sumę średnich arytmetycznych po kolejnym
uśrednieniu przez 20
            std::cout << wektor[wektor.size()-1] << std::endl;
        }
    } while (liczba > 0);

    // wyświetl liczbę, która spowodowała wysypanie programu
    std::list<int>::iterator iterator_listy = lista.begin();

    std::cout << *iterator_listy << std::endl;
}

```

#### TEST KODU:

```

int main(){
    sample_stl_program();

    system("pause");
    return 0;
}

```

Test polegał na podaniu danych klawiatury.

#### WNIOSKI:

Treścią zadania było, żeby napisać kod korzystający z metod, które daje nam biblioteka Vector i List nie miał on mieć sensu, ani robić nic sensownego. Fajnym dodatkiem zamiast warunku czy liczba z listy jest równa np. 7 jest metoda `remove_if()`, do której potrzebna jest wartość bool'owska np. jakaś funkcja która sprawdza nam czy liczba jest parzysta czy nie i zwracająca wartość `true` lub `false`.

## Zadanie 4

### OPIS:

Poleceniem było napisanie funkcji, która wypisze nam na wyjście `std::cout` zawartość tablicy bądź pojemnika STL. Ponadto funkcja miała mieć nazwę `wypisz_na_cout()` dlatego w kodzie zastosowane są templaty. Każda linia kodu ma do siebie komentarz opisujący jej rolę.

### KOD:

```
// Przypadek gdy podajemy np adres na pierwszy element w tablicy i ostatni
template <class Iter>
void wypisz_na_cout(Iter begin, Iter end){
    //std::cout << "dsadsa" << std::endl;
    // wypisujemy na wyjście
    while (begin != end){
        // wypisz wartosc znajdującą się pod danym adresem
        std::cout << *begin << ' ';
        // zwiększ o jeden adres, w celu dostania się do kolejnego elementu
        tablicy
        *(begin++);
    }
    std::cout << std::endl;
}

// Przypadek gdy podajemy adres na pierwszy element w tablicy i podajemy jej dlugosc
jako int
template <class Iter>
void wypisz_na_cout(Iter begin, int len){
    while (len > 0){
        // wypisz wartosc znajdującą się pod danym adresem
        std::cout << *begin;
        // zwiększ o jeden adres, w celu dostania się do kolejnego elementu
        tablicy
        *(begin++);
        // jeżeli pętla trwa dłużej niż raz zrób biały znak przed kolejnym
        if (len > 1){
            std::cout << ' ';
        }
        len--;
    }
    std::cout << std::endl;
}

// przypadek gdy podajemy bezpośrednio wektor
template <typename Container>
void wypisz_na_cout(const Container& c){
    // pobierz iterator z klasy, z którą podajemy np. podamy vector,
    // to będzie to std::vector<int>::iterator nazwa_itr = c.begin(),
    // begin() to metoda z klasy vector
    typename Container::const_iterator begin = c.begin();
    typename Container::const_iterator end = c.end();

    while (begin != end){
        std::cout << *begin << ' ';
        ++begin;
    }
    std::cout << std::endl;
}
```

## TEST KODU:

```
int main(){  
  
    //sample_stl_program();  
  
    int t[4];  
    std::vector<int> v(4);  
  
    for (int i = 0; i < 4; ++i) {  
        t[i] = i + 1;  
        v[i] = i + 1;  
    }  
  
    wypisz_na_cout(t, t + 4);  
    wypisz_na_cout(t, 4);  
  
    wypisz_na_cout(v.begin(), v.end());  
    wypisz_na_cout(v.begin(), 4);  
    wypisz_na_cout(v);  
  
    system("pause");  
    return 0;  
}
```

## WNIOSKI:

Kod jest napisany zgodnie z zaleceniami. Jedyne co można by było poprawić lub usprawnić, to napisać unit testy i sprawdzić czy dla różnych przypadków różnych pojemników typu np. List czy inne działałyby poprawnie.

## Zadanie 5

### OPIS:

Zadanie polegało na napisaniu szablonu przechodzącego przez pewien ciąg elementów i jeżeli zostanie spełniony jakiś dany przez nas warunek, to ma wykonać się jakaś czynność.

Poszczególne fragmenty kodu są opisane komentarzami.

### KOD:

```
template <typename Iter, typename Cond, typename Oper>  
void wykonaj_na_spełniających(Iter begin, Iter end, Cond cnd, Oper op)  
{  
    // dopóki iterator początku nie równa się końcowemu to wykonuj pętlę  
    while (begin != end) {  
        // sprawdzamy czy warunek jest prawdą  
        // i jeżeli tak to wywołujemy daną funkcjonalność  
        // i zaczynamy od pierwszej wartości  
        if (cnd(*begin)){  
            op(*begin);  
        }  
        // przejdź do kolejnego elementu  
        ++begin;  
    }  
}  
  
struct Zdolność_kredytowa{  
    bool operator() (int x){
```

```

        return x >= 60000;
    }
};

struct Przelej_300_kola{
    int& przelew;

    // konstruktor struktury przelej_300_kola
    Przelej_300_kola(int& bank) : przelew(bank)
    {

    }

    void operator() (int& x){
        // dopsiuj do zmiennej dodatkowe 300k jesli warunek bool operation() ==
true
        x += 300000;
        przelew += 300000;
    }
};

```

#### TEST KODU:

```

int main(){

    int konta[] = { 10123, 50, 999000, 100, 500, 60000, 100000 };
    int ilosc_kont = sizeof(konta) / sizeof(konta[0]);

    std::cout << "Konta ubiegające się o pożyczkę: ";
    wypisz_na_cout(konta, konta + ilosc_kont);

    int ilosc_pożyczonych_pieniedzy = 0;
    Przelej_300_kola pożyczka(ilosc_pożyczonych_pieniedzy);

    wykonaj_na_spełniających(konta, konta + ilosc_kont, Zdolność_kredytowa(),
pożyczka);

    std::cout << "Konta po przyznaniu pożyczki: ";
    wypisz_na_cout(konta, konta + ilosc_kont);
    std::cout << "Ile pożyczono: " << ilosc_pożyczonych_pieniedzy << '\n';

    system("pause");
    return 0;
}

```

#### WNIOSKI:

Kod spełnia swoje założenia. Ciekawą rzeczą jaką można było by zrobić to np. zaimplementować dokładniejszy szereg metod sprawdzających zdolność kredytową osoby, jej historie itp.

#### Zadanie 6

##### OPIS:

Polecenie zadania to napisać własną implementację klasy List o nazwie Lista. Należy wykorzystać iteratory. Dodatkowo dla przykładu napisać jedną metodę klasy List dodającą element i begin() i end(). Co i za co jest odpowiedzialne jest udokumentowane w kodzie komentarzami.

##### KOD:

```

#ifndef LISTA_H
#define LISTA_H

#include<iostream>
#include <cstdlib>
#include<iterator>

template <typename T>
class Lista{
private:
    // Klasy definiujemy w klasie ze wzgledu na takową implementacje w bibliotece
    List
    // tam tak samo korzystamy np list<int>::iterator name = na co wskazuje

    // zdefiniowane po to, żeby można było zadeklarować że jest to klasa
    // zaprzyjaźniona z klasą Element
    class Iterator;

    class Element{
        T data;
        Element* next;

        Element(const T& val, Element* nxt = 0) : data(val), next(nxt)
        {

        }

        friend class Lista;
        friend class Iterator;
    };

    class Iterator{
        Element* current;

        Iterator(Element* position) : current(position)
        {

        }

        friend class Lista;

    public:
        // można zrobić typdefy dla ułatwienia
        Iterator() : current(0)
        {

        }

        // konstruktory kopiujace same zostaną wygenerowane przez kompilator
        // definiujemy operatory przeładowania

        // Jak ma zachowywać się gdy poprosimy, żeby zwrócił nam na co wskazuje
        obiekt
        T& operator*(){
            return current->data;
        }

        // przeładowanie wywołania na metody danego obiektu
        T* operator->(){
            return &(current->data);
        }
    };
};

```

```

    }
    // preinkrementacja
    T& operator++(){
        current = current->next;
        return *this;
    }

```

```

    // postinkrementacja
    T& operator++(int){
        Iterator itr = *this;
        current = current->next;
        return itr;
    }
    // predekrementacja
    T& operator--(){
        current = current->prev;
        return *this;
    }

```

```

    // postdekrementacja
    T& operator--(int){
        Iterator itr = *this;
        current = current->prev;
        return itr;
    }

```

```

    // operacja przyrównania dwóch iteratorów
    bool operator==(const Iterator itr){
        // czy current, który podajemy równa się z curentem który mamy
        return current == itr.current;
    }

```

```

    bool operator!=(const Iterator itr){
        // czy current, który podajemy NIE równa się z curentem który mamy
        return current != itr.current;
    }
};

```

```

    Element* head;

```

```

public:

```

```

    Lista() : head(0)
    {

```

```

    }

```

```

    ~Lista();

```

```

    // definiujemy dla przykładu jedną z metod biblioteki lista np. pop_
    void push_front(const T&);
    // push back czy musze zdefiniować zmienna tail lub jakąś end?
    void push_back(const T&);
    // ustaw iterator na pierwszy element w tablicy
    void begin(){
        return Iterator(head);
    }
    // ustaw iterator na ostatni element w pojemniku
    void end(){
        return Iterator(0);
    }

```

```

};
#endif

// destruktor
template<typename T>
Lista<T>::~~Lista(){
    // tworzymy wskaźnik na obiekt typu Element
    Element* p;

    // dopóki istnieje head
    // usuwaj element na który wskazuje wskaźnik i przechodź do następnego
    while(head){
        p = head;
        head = head->next;
        delete p;
    }
}

template <typename T>
void Lista<T>::push_front(const T& value){
    // stwórz nową tablicę typu Element, z wartościami value i o zadanej długości
    Element* p = new Element(value, head);
    // ustaw wskaźnik head na p
    head = p;
}

```

#### TEST KODU:

```

int main(){

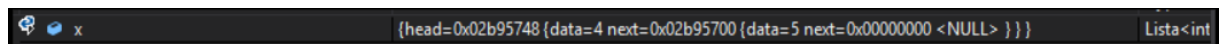
    Lista<int> x;

    x.push_front(5);
    x.push_front(4);

    system("pause");
    return 0;
}

```

Po użyciu DEBUGGERA widać, że metoda działa poprawnie.



#### WNIOSKI:

Klasa spełnia swoje założenia. Można by było dodać więcej metod i sprawdzić czy na pewno przewidziane zostały wszystkie scenariusze.

#### Zadanie 7

##### OPIS:

Zadanie polegało na napisaniu testu przy użyciu zewnętrznej biblioteki boost.

Test sprawdza poprawność iteratora z zadania 6, a dokładniej jego wymagania odnośnie konceptu ForwardIterator.

##### KOD:



```

#include <boost/concept_check.hpp>

int main(){

    // zadanie 7

    boost::function_requires< ForwardIteratorConcept<vector<double>::iterator> >()>;

    // typedef zeby kod był czytelniejszy i nie trzeba było pisac w srodku
    function_requires <Lista<char>::iterator>
    // tylko po prostu it_listy
    typedef Lista<char>::iterator it_listy;
    boost::function_requires< ForwardIteratorConcept<it_listy> >()>;

    std::cout << "Wszystko OK.\n";

    system("pause");
    return 0;
}

```

#### WNIOSKI:

Kod działa poprawnie. Co do wniosków, to jest to tylko test do zadania 6, i jedynym problemem była instalacja biblioteki boost i zapoznanie się z jej dokumentacją.