|  |
| --- |
| **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 liczbe 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 liczbe do listy

if (liczba % 2 == 0){

lista.push\_front(liczba);

}

else lista.push\_back(liczba);

// Jeżeli lista > 10 to zrob sume z tych liczb i przekaz ją do wektora

if (lista.size() >= 10){

std::cout << lista.size() << " danych zostalo 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;

}

// wyczyść liste

lista.clear();

// zapisujemy do wektora sume 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 liczbe na wyjscie

std::cout << wektor.front() << std::endl;

// gdzy wektor ma wielkosc 20 sciagamy 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 sume sredniech arytmetycznych po kolejnym uśrednieniu przez 20

std::cout << wektor[wektor.size()-1] << std::endl;

}

}

} while (liczba > 0);

// wyświetl liczbe, 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 zwaracająca 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 role.

**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 << ' ';

// zwieksz o jeden adres, w celu dostania sie 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;

// zwieksz o jeden adres, w celu dostania sie 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 bezposrednio 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 działałby 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\_spelniajacych(Iter begin, Iter end, Cond cnd, Oper op)

{

// dopóki iterator poczatku nie rowna sie koncowemu to wykonuj petle

while (begin != end) {

// sprawdzamy czy prawdą jest

// i jeżeli tak to zacznik od pierwszej wartości

if (cnd(\*begin)){

op(\*begin);

}

// przejdz 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 pożyczke: ";

wypisz\_na\_cout(konta, konta + ilosc\_kont);

int ilosc\_pozyczonych\_pieniedzy = 0;

Przelej\_300\_kola pozyczka(ilosc\_pozyczonych\_pieniedzy);

wykonaj\_na\_spelniajacych(konta, konta + ilosc\_kont, Zdolność\_kredytowa(), pozyczka);

std::cout << "Konta po przyznaniu pozyczki: ";

wypisz\_na\_cout(konta, konta + ilosc\_kont);

std::cout << "Ile pożyczono: " << ilosc\_pozyczonych\_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. Jest to doby sposób implementacji niektórych funkcji dla np. ATM.

**Zadanie 6**

**OPIS:**

**KOD:**

**WNIOSKI:**

**Zadanie 7**

**OPIS:**

**KOD:**

**WNIOSKI:**