Programowanie obiektowe wykład 1a

PODSTAWY PROGRAMOWANIA OBIEKTOWEGO W C++

Obiekt

Obiekty programowe to zbiór własności i zachowań (zmiennych i metod). Podobnie jak w świecie rzeczywistym obiekty posiadają swój stan i zachowanie.

Komunikat

Wszystkie informacje przekazywane do obiektu to komunikaty. Komunikat można traktować również jako parametr wywołania wybranej metody.

Klasa

Klasa to prototyp obiektu. Podobnie jak zmienne są określonych typów tak obiekty są instancjami klas. Klasę można traktować jako swego rodzaju szablon obiektu/ów. Atrybuty (zmienne) i metody (funkcje) w klasie. Konkretne obiekty to *instancje* klas. (Trochę jak Typy i zmienne w C)
Klasa jest abstrakcyjna jeżeli nie można utworzyć jej instancji. W języku C++ dzieje się tak, gdy jedna z metod jest zadeklarowana jako metoda czysto wirtualna (wirtualna bez implementacji)
Klasa jest finalna jeżeli nie można od niej utworzyć klasy potomnej.

Definiowanie klasy

```
class MojaKlasa
{
public:
      MojaKlasa();
      virtual ~MojaKlasa();
      virtual void DrukujNazweKlasy();
      void DrukujNazweKlasy2();
private:
      int zmienna_prywatna;
};
MojaKlasa::MojaKlasa()
{
};
MojaKlasa::~MojaKlasa()
{};
void MojaKlasa::DrukujNazweKlasy()
{
  cout<<"To jest klasa pierwsza"<<endl;
};
void MojaKlasa::DrukujNazweKlasy2()
{
  cout<<"To jest klasa pierwsza"<<endl;
```

```
};
```

W definicji klasy wyróżniamy trzy bloki public , protected i private – p. punkt Enkapsulacja

Funkcje i zmienne składowe const

Funkcje i składowe, które nie zmieniają się w całej klasie można należy zadeklarować jako *const*.

Funkcje wirtualne

Funkcje (metody), które są zdefiniowane w klasie bazowej i mogą być redefiniowane w klasach potomnych w taki sposób, że nawet jeżeli konkretna instancja klasy jest wskazywana przez wskaźniki do klasy bazowej kompilator wie, że ma być użyta metoda z klasy potomnej.

```
#include <iostream>
using namespace std;
class MojaKlasa
{
public:
      MojaKlasa();
      virtual ~MojaKlasa();
      virtual void DrukujNazweKlasy();
      void DrukujNazweKlasy2();
};

MojaKlasa::MojaKlasa()
{
};


MojaKlasa::~MojaKlasa()
{};

void MojaKlasa::DrukujNazweKlasy()
{
  cout<<"To jest klasa pierwsza"<<endl;
};
void MojaKlasa::DrukujNazweKlasy2()
{
  cout<<"To jest klasa pierwsza"<<endl;
};


class MojaKlasaPotomna : public MojaKlasa
{
public:
      MojaKlasaPotomna();
      ~MojaKlasaPotomna();
      void DrukujNazweKlasy();
      void DrukujNazweKlasy2();
};

MojaKlasaPotomna::MojaKlasaPotomna()
{
};
```

```
MojaKlasaPotomna  ::~MojaKlasaPotomna()
{};

void MojaKlasaPotomna::DrukujNazweKlasy()
{
 cout << "To jest klasa druga" << endl;
};

void MojaKlasaPotomna::DrukujNazweKlasy2()
{
 cout << "To jest klasa druga" << endl;
};

int main()
{

  int i;
  MojaKlasa * mk =new MojaKlasaPotomna();
  mk->DrukujNazweKlasy();
  mk->DrukujNazweKlasy2();
  cin>>i;
  return 0;
}
```

Dziedziczenie i funkcje wirtualne umożliwiają mechanizm *Polimorfizmu*, czyli wykazywanie przez metodę różnych form działania w zależności od tego jaki typ obiektu jest wskazywany przez wskaźnik lub referencję.

Zmienna wskaźnikowa this:

- Zmiena wskaźnikowa *this* wskazuje na adres instancji klasy (konkretnego obiektu)
- Zmienna *this* nie jest dostępna dla metod statycznych
- Zmienna *this* nie może być modyfikowana.

Konstruktor/ destruktor

Metoda/y o nazwie takiej jak klasa, wywoływana przy inicjalizacji/usuwaniu obiektu danej klasy
Klasy posiadają specjalne funkcje składowe, zwane konstruktorami. Konstruktor (ang. constructor) może w razie potrzeby posiadać parametry, ale nie może zwracać wartości — nawet typu void. Konstruktor jest metodą klasy o takiej samej nazwie, jak nazwa klasy.
Konstruktor tworzy i inicjalizuje obiekt danej klasy, zaś destruktor porządkuje obiekt i zwalnia zaalokowaną pamięć. Destruktor zawsze nosi nazwę klasy, poprzedzoną znakiem tyldy (~). Destruktory nie mają argumentów i nie zwracają wartości.

Konstruktor kopiujący

Oprócz domyślnego konstruktora i destruktora, kompilator dostarcza także domyślnego konstruktora kopiującego. Konstruktor kopiujący jest wywoływany za każdym razem, gdy tworzona jest kopia obiektu.
Gdy w programie jest przekazywany obiekt przez wartość, czy to jako parametr funkcji czy też jako jej wartość zwracaną , tworzona jest tymczasowa kopia tego obiektu. Jeśli obiekt jest

obiektem zdefiniowanym przez użytkownika, wywoływany jest konstruktor kopiujący danej klasy
Wszystkie konstruktory kopiujące posiadają jeden parametr; jest nim referencja do obiektu tej samej klasy. Dobrym pomysłem jest oznaczenie tej referencji jako const, gdyż wtedy konstruktor nie ma możliwości modyfikacji otrzymanego obiektu. Na przykład:

```
CAT(const CAT & theCat);
```

W tym przypadku konstruktor CAT otrzymuje stałą referencję do istniejącego obiektu klasy CAT. Celem konstruktora kopiującego jest utworzenie kopii obiektu theCat.
Domyślny konstruktor kopiujący po prostu kopiuje każdą zmienną składową z obiektu otrzymanego jako parametr do odpowiedniej zmiennej składowej obiektu tymczasowego.

Przeciążanie operatorów

C++ posiada liczne typy wbudowane, takie jak int, float, char, itd. Każdy z nich posiada własne wbudowane operatory, takie jak dodawanie (+) czy mnożenie (*). C++ umożliwia stworzenie takich operatorów także dla klas definiowanych przez użytkownika.

Enkapsulacja

Nie wszystkie zmienne i metody obiektu są widoczne na zewnątrz. Część z nich jest dostępne tylko wewnątrz klasy, w której jest zdefiniowana lub tylko wewnątrz podklas. Ten proces nosi nazwę enkapsulacji i do jego implementacji używa się następujących modyfikatorów zmiennych:

Specyfikator	Klasa	Podklasa	Świat
private	X		
protected	X	X*	
public	X	X	X

Dziedziczenie (Inheritance)

Funkcjonalność każda klasa może zostać rozszerzona poprzez zdefiniowanie dla niej klasy pochodnej. Klasa pochodna dziedziczy od klasy nadrzędnej wszystkie zmienne i metody a ponadto może sama definiować (nadpisywać) metody klasy nadrzędnej i/lub zawierać własne metody i zmienne. Mechanizm dziedziczenia i hierarchii klas jest jednym z najbardziej istotnych elementów programowania obiektowego (np. MFC w C++).

Dziedziczenie to mechanizm przekazywania klasom pochodnym wszystkich (nie zadeklarowanych jako prywatne) zmiennych i funkcji zdefiniowanych w klasie pierwotnej.

Podczas dziedziczenia zawartość klasy podstawowej staje się automatycznie zawartością klasy pochodnej. Zatem wszystkie składniki i funkcje składowe stają się dostępne w klasie odziedziczonej. Oczywiście ta *dostępność* jest uwarunkowana sposobem dziedziczenia. Dziedziczenie publiczne jest najprostsze (domyślne) i chyba najczęściej stosowane. Praktycznie

nie powoduje żadnych zmian. Dostęp do składników odziedziczonych jest nadal taki sam, jak w klasie podstawowej. Możliwe jest jednak zdefiniowanie innych sposobów dziedziczenia

składniki klasy podstawowej	sposób dziedziczenia	składniki klasy pochodnej
składniki prywatne składniki chronione składniki publiczne	Prywatne	składniki prywatne
składniki prywatne składniki chronione składniki publiczne	Chronione	składniki prywatne składniki chronione składniki chronione
składniki prywatne składniki chronione składniki publiczne	Publiczne	składniki prywatne składniki chronione składniki publiczne

Zmienne instancji i zmienne klas

Domyślnie wszystkie zmienne są zmiennymi instancji, czyli dotyczą obiektów, w których są wywoływane. Może się jednak zdarzyć sytuacja, gdy chcemy, aby wartość zmiennej była dzielona pomiędzy wszystkie instancje danej klasy. Możemy wtedy zadeklarować zmienną klasy używając słowa kluczowego *static*

```
class MojaKlasa
{
public:
        MojaKlasa();
        ~MojaKlasa();
        virtual void DrukujNazweKlasy();
        void DrukujNazweKlasy2();
        int static const t=5 ;
};

int main()
{
  cout<<MojaKlasa::t;
  return 0;
}
```

Napisz własną klasę *MojLancuch*, która zdefiniuje operator – dla łańcuchów wykonujący „odejmowanie” w następujący sposób:
Jeśli łańcuch a zaczyna się od b to a-b jest częścią łańcucha występującą po b. Np „architektura”-„archi” = „tektura”
Zmodyfikuj klasę, tak aby uogólnić operator odejmowania („alabaster” – „bas” = „alater” itp)

PRZEGLĄD NAJWAŻNIEJSZYCH BIBLIOTEK

Strings

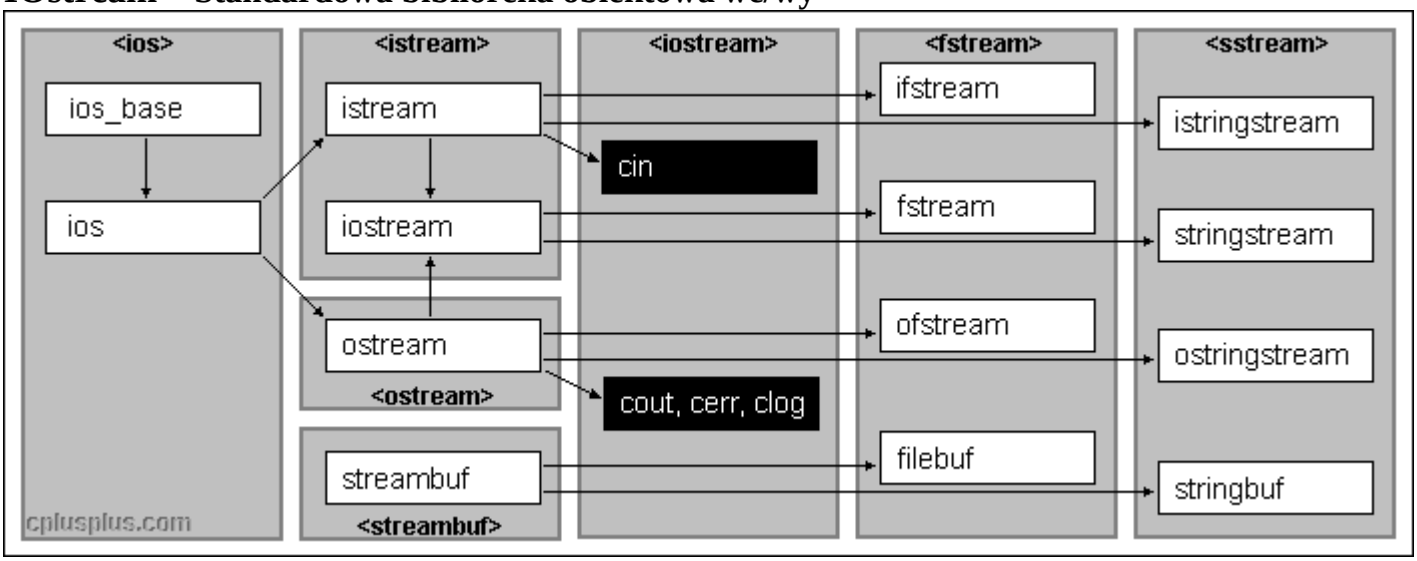
Biblioteka zmiennych tekstowych typu String w **C++**

operator+	Konkatenacja (funkcja)
swap	Wymiana zawartości dwóch łańcuchów(funkcja)
operatory porównania	Porównanie zawartości łańcuchów w kolejności alfabetycznej (funkcja)

Łańuchy i strumienie:

getline	Odczytanie pojedynczej linii ze strumienia(funkcja)
operator<<	Wstawienie łańcucha do strumienia(funkcja)
operator>>	Odczytanie łańcucha ze strumienia(funkcja)

IOstream – Standardowa biblioreka obiektowa we/wy



Biblioteka *iostream* służy do zapewnienia dostępu wejścia / wyjścia wykorzystując koncepcję strumieni. Strumień może być połączony z plikiem, klawiaturą konsolą i innymi fizycznymi źródłami.

Obiekty standardowe

cin, cout, cerr czyli standardowe strumienie: wejściowy, wyjściowy i błędu oraz ich „szerokie” (dwubajtowe) odpowiedniki *wcin, wcout, wcerr* i *wclog*.

streamoff streamsize - pozycja i rozmiar strumienia

Manipulatory

Manipulatory to funkcje globalne przewidziane do użycia z (<<) i (>>) np: *endl, hex , scientific*

Hierarchia

- <ios>, <istream>, <ostream>, <streambuf> <iosfwd> klasy bazowe
- <iostream> strumienie standardowe wejścia-wyjścia (np cin cout).

- <fstream> strumień związany z plikiem
- <sstream>: manipulacja obiektami STL string tak jakby były strumieniami.
- <iomanip> standardowe maniplatory

KLASY:

ios_base	Klasa bazowa dla strumieniu zależnych od typów (klasa)
ios	Klasa bazowa dla strumieniu zależnych od typów (klasa)
istream	Input stream (klasa)
ostream	Output Stream (klasa)
iostream	Input/Output Stream (klasa)
ifstream	Input file stream class (klasa)
ofstream	Output file stream (klasa)
fstream	Input/output file stream class (klasa)
istringstream	Input string stream class (klasa)
ostringstream	Output string stream class (klasa)
stringstream	Input/output string stream class (klasa)
streambuf	Base buffer class for streams (klasa)
filebuf	File stream buffer (klasa)
stringbuf	String stream buffer (klasa)

Obiekty:

cin	Standardowy strumień wejściowy (obiekt)
cout	Standardowy strumień wyjściowy (obiekt)
cerr	Standardowy strumień (obiekt)
clog	Standardowy strumień plików dziennika (logów)(obiekt)

Typy:

fpos	Aktualna wartość wskaźnika do pliku (szablon klasy)

Manipulatorsy:

dec	liczba dziesiętna (manipulator funkcji)
endl	Koniec linii (manipulator funkcji)
ends	Spacja (manipulator funkcji)
fixed	Notacja z ustaloną kropką dziesiętną (manipulator funkcji)
flush	
hex	Liczba heksadecymalna (manipulator funkcji)
internal	
left	Wyrównanie do lewej (manipulator funkcji)
noboolalpha	
noshowbase	
noshowpoint	
noshowpos	
noskipws	
nounitbuf	
nouppercase	
oct	liczba ósemkowa (manipulator funkcji)
resetiosflags	
right	Wyrównanie do prawej (manipulator funkcji)

scientific	Format liczb naukowy (manipulator funkcji)
setbase	
setfill	
setiosflags	
setprecision	
setw	
showbase	
showpoint	
showpos	
skipws	
unitbuf	
uppercase	Wielkie litery
ws	

Standard Template Library: Kontenery

Kontener przechowuje kolekcję obiektów (elementów). Kontenery są zaimplementowane jako szablony klasy, dzięki czemu mogą w bardzo elastyczny sposób operować na różnych typach elementów. Kontenery same organizują przestrzeń do zapamiętania elementów oraz udostępniają dostęp do nich przez funkcje oraz przez *iteratory*(obiekty referencyjne analogiczne do zmiennych wskaźnikowych)

Kontenery -Szablony klas

Kontenery sekwencyjne:

vector	Wektor (szablon klasy)
deque	Kolejka dwóstronna (szablon klasy)
list	Lista (szablon klasy)

Adaptery kontenerów (funkcjonalność do uzupełnienia):

stack	LIFO stos(szablon klasy)
queue	FIFO kolejka (szablon klasy)
priority_queue	Kolejka priorytetowa (szablon klasy)

Kontenry asocjacyjne:

set	Zbiór (szablon klasy)
multiset	Zbiór z wieloma kluczami (szablon klasy)
map	Mapa (szablon klasy)
multimap	Mapa z wieloma kluczami (szablon klasy)
bitset	Klucz bitowy (szablon klasy)

		Kontenery sekwencyjne				**Kontenery asocjacyjne**			
Plik nagłówkowy		**<vector>**	**<deque>**		**<list>**	**<set>**		**<map>**	**<bitset>**
Zawiera:		**vector**	**deque**	**list**	**set**	**multiset**	**map**	**multimap**	**bitset**
	Konstruktor	constructor	constructor	constructor	constructor	constructor	constructor	constructor	constructor
	Destruktor	destructor	destructor	destructor	destructor	destructor	destructor	destructor	

	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operators
Iteratory	Początek	begin	begin	begin	begin	begin	begin	begin	
	koniec	end	end	end	end	end	end	end	
	Początek odwrotny	rbegin	rbegin	rbegin	rbegin	rbegin	rbegin	rbegin	
	Koniec odwrotny	rend	rend	rend	rend	rend	rend	rend	
Rozmiar	Rozmiar	size	size	size	size	size	size	size	size
	Maksymalny rozmiar	max_size	max_size	max_size	max_size	max_size	max_size	max_size	
	Czy pusty	empty	empty	empty	empty	empty	empty	empty	
	Zmień rozmiar	resize	resize	resize					
Dostęp do elementów	Początek	front	front	front					
	Koniec	back	back	back					
	operator[]	operator[]	operator[]				operator[]		operator[]
	Na pozycji	at	at						
Modyfikujące	Przypisz	assign	assign	assign					
	Wstaw	insert	insert	insert	insert	insert	insert	insert	
	Skasuj	erase	erase	erase	erase	erase	erase	erase	
	Zamień	swap	swap	swap	swap	swap	swap	swap	
	Wyczyść wszystko	clear	clear	clear	clear	clear	clear	clear	
	Dopisz na początek		push_front	push_front					
	Pobierz z początku		pop_front	pop_front					
	Dopisz na koniec	push_back	push_back	push_back					
	Pobierz z z końca	pop_back	pop_back	pop_back					
Obserwacja	Porównanie kluczy				key_comp	key_comp	key_comp	key_comp	
	Porównanie wartości				value_comp	value_comp	value_comp	value_comp	
Operacje	Znajdź				find	find	find	find	

	Policz				count	count	count	count	count
	Dolny zakres				lower_bound	lower_bound	lower_bound	lower_bound	
	Górny zakres				upper_bound	upper_bound	upper_bound	upper_bound	
	Równe zakresy				equal_range	equal_range	equal_range	equal_range	

Adaptery kontenerów

			Adaptery kontenerów		
Nagłówki			**<stack>**	**<queue>**	
Zawiera			**stack**	**queue**	**priority_queue**
	konstruktor	*	constructor	constructor	constructor
Wielkość	Rozmiar	O(1)	size	size	size
	Czy pusty	O(1)	empty	empty	empty
Dostęp do elementów	Początek	O(1)		front	
	Koniec	O(1)		back	
	Góra (stos)	O(1)	top		top
Modyfikatory	Push	O(1)	push	push	push
	Pop	O(1)	pop	pop	pop

STL Algorytmy

Standard Template Library: Algorytmy

Funkcje

Funkcje nie modyfikujące kolejności:

for_each	Dla każdego z zakresu (szablon funkcji)
find	Znajdź wartość w zakresie(szablon funkcji)
find_if	Znajdź element w zakresie (szablon funkcji)
find_end	Znajdź ostatnie wystąpienie w zakresie (szablon funkcji)
find_first_of	Znajdź ostatnie wystąpienie w zakresie (szablon funkcji)
adjacent_find	Znajdź odpowiednie przylegające elementy w zakresie (szablon funkcji)
count	Policz elementy w zakresie (szablon funkcji)
count_if	Policz elementy w zakresie spełniające warunek(szablon funkcji)
mismatch	Pierwsza pozycja, gdzie dwie listy się różnią (szablon funkcji)
equal	Sprawdzenie czy dwa zakresy elementów zawierają takie same elementy(szablon funkcji)
search	Znalezienie podlisty w zakresie (liście) (szablon funkcji)
search_n	Znalezienie podlisty w zakresie (liście) na n-tej pozycji (szablon funkcji)

Funkcje Modyfikujące kolejność:

copy	Kopiowanie zakresu elementów (szablon funkcji)
copy_backward	Kopiowanie zakresu elementów (szablon funkcji)
swap	Wymiana dwóch elementów(szablon funkcji)
swap_ranges	Wymiana dwóch zakresów (szablon funkcji)
iter_swap	Wymiana wartości wskazywanych przez dwa iteratory (szablon funkcji)
transform	Zastosowanie funkcji dla zakresu (szablon funkcji)
replace	Wymiana wartości w zakresie (szablon funkcji)
replace_if	Wymiana warunkowa wartości w zakresie (szablon funkcji)
replace_copy	Kopiowanie i wymiana wartości w zakresie (szablon funkcji)
replace_copy_if	Kopiowanie i wymiana warunkoa wartości w zakresie (szablon funkcji)
fill	Wypełnienie zakresu wartością (szablon funkcji)
fill_n	Wypełnienie sekwencji wartością (szablon funkcji)
generate	Wypełnienie zakresu wartością wygenerowaną funkcją (szablon funkcji)
generate_n	Wypełnienie sekwencji wartością wygenerowaną funkcją (szablon funkcji)
remove	Usunięcie wartości z zakresu (szablon funkcji)
remove_if	Usunięcie warunkowe wartości z zakresu (szablon funkcji)
remove_copy	Copy range removing value (szablon funkcji)
remove_copy_if	Copy range removing values (szablon funkcji)
unique	Usunięcie powtórzeń (szablon funkcji)
unique_copy	Kopiowanie zakresu z usuniętymi powtórzeniami (szablon funkcji)
reverse	Odwrotna kolejność w zakresie (szablon funkcji)
reverse_copy	Przekopiowanie w odwrotnej kolejnośc (szablon funkcji)
rotate	Rotate elements in range (szablon funkcji)
rotate_copy	Copy rotated range (szablon funkcji)
random_shuffle	Ułożenie losowe elementów (szablon funkcji)
partition	Podział kontenera (szablon funkcji)
stable_partition	Podział kontenera - stabilny porządek(szablon funkcji)

Sortowanie:

sort	Sortowanie elementów w zakresie (szablon funkcji)
stable_sort	Sort elements preserving order of equivalents (szablon funkcji)
partial_sort	Partially Sort elements in range (szablon funkcji)
partial_sort_copy	Copy and partially sort range (szablon funkcji)
nth_element	n-ty element (szablon funkcji)

Przeszukiwanie binarne (Na posortowanych elementach):

lower_bound	Return iterator to lower bound (szablon funkcji)
upper_bound	Return iterator to upper bound (szablon funkcji)
equal_range	Get subrange of equal elements (szablon funkcji)
binary_search	Test if value exists in sorted array (szablon funkcji)

Łączenie (operating on sorted ranges):

merge	Połączenie dwóch kolekcji (zakresów) (szablon funkcji)
inplace_merge	Połączenie kilku kolekcji (szablon funkcji)

includes	Sprawdzenie czy zakres zawiera w sobie inny zakres(szablon funkcji)
set_union	Unia dwóch zakresów (kolekcji) (szablon funkcji)
set_intersection	Część wspólna (szablon funkcji)
set_difference	Różnica dwóch kolekcji (szablon funkcji)
set_symmetric_difference	Różnica symetryczna dwóch kolekcji (szablon funkcji)

Operacje na stercie:

push_heap	Położenie na stercie (szablon funkcji)
pop_heap	Pobranie ze sterty
make_heap	"Kopcowanie" (szablon funkcji)
sort_heap	Sort elements of heap (szablon funkcji)

Min/max:

min	Return the lesser of two arguments (szablon funkcji)
max	Return the greater of two arguments (szablon funkcji)
min_element	Return smallest element in range (szablon funkcji)
max_element	Return largest element in range (szablon funkcji)
lexicographical_compare	Lexicographical less-than comparison (szablon funkcji)
next_permutation	Transform range to next permutation (szablon funkcji)
prev_permutation	Transform range to previous permutation (szablon funkcji)