```
/* Wirtualny konstruktor kopiujący*/

#include<iostream>
using namespace std;

class Ssak
{
public:
	Ssak(int w):waga(w)
	{cout<<"Konstruktor klasy Ssak...\n";   }
	Ssak (Ssak& ref);
	virtual void Glos(){cout<<"Glos ssaka\n";}
	virtual Ssak* Klonuj()
	{return new Ssak(*this);}
	int Ile_wazy()
	{return this->waga;}
protected:
	int waga;
};

Ssak::Ssak(Ssak& ref):waga(ref.Ile_wazy())
{cout << "Konstruktor kopiujacy klasy Ssak...\n";}
```

```
class Pies : public Ssak
{
public:
	Pies():Ssak(10)
	{cout<<"Konstruktor klasy Pies...\n";}
	Pies(Pies & ref);
	virtual Ssak* Klonuj()
	{return new Pies(*this);}
	void Glos(){cout<<"Hau\n";}
};
Pies::Pies(Pies& ref):Ssak(ref)
{cout<<"Konstruktor kopiujacy klasy Pies...\n";}

class Kot : public Ssak
{
public:
	Kot():Ssak(2)
	{cout<<"Konstruktor klasy Kot...\n";}
	Kot (Kot& ref);
	virtual Ssak* Klonuj()
	{return new Kot(*this);}
	void Glos(){cout<<"Miau\n";}
};
```

```
Kot::Kot(Kot& ref):Ssak(ref)
{cout<<"Konstruktor kopiujacy klasy Kot...\n";}

enum ZWIERZAKI {SSAK, KOT, PIES};

void main()
{
	Ssak *tablica [3];
	Ssak* ptr;
	int wybor;
	for  (int i = 0; i < 3; i++)
	{
		cout<<"(1)pies	(2)kot (3)ssak : ";
		cin>>wybor;
		switch (wybor)
		{
		case PIES: ptr = new Pies;
			break;
		case KOT: ptr = new Kot;
			break;
		default: ptr = new Ssak(1);
			break;
		}
```

```
            tablica[i]=ptr;
        }

        Ssak *tablica1[3];

        for  (int i = 0; i < 3 ; i++)
        {
            tablica[i]->Glos();
            tablica1[i]=tablica[i]->Klonuj();
        }

        cout<<"Po sklonowaniu :\n\n";
        for  (int i = 0; i < 3; i++)
        {
            cout<<"tablica1["<<i<<"]\n";
            tablica1[i]->Glos();
            cout<<"Waga : "<<tablica1[i]->Ile_wazy()<<endl;
        }
        system("pause");
}
```

```
(1)pies (2)kot  (3)ssak : 1
Konstruktor klasy Ssak...
Konstruktor klasy Kot...
(1)pies (2)kot  (3)ssak : 2
Konstruktor klasy Ssak...
Konstruktor klasy Pies...
(1)pies (2)kot  (3)ssak : 3
Konstruktor klasy Ssak...
Miau
Konstruktor kopiujacy klasy Ssak...
Konstruktor kopiujacy klasy Kot...
Hau
Konstruktor kopiujacy klasy Ssak...
Konstruktor kopiujacy klasy Pies...
Glos ssaka
Konstruktor kopiujacy klasy Ssak...
```

```
Po sklonowaniu :

tablica1[0]
Miau
Waga : 2
tablica1[1]
Hau
Waga : 10
tablica1[2]
Glos ssaka
Waga : 1
Press any key to continue . . .
```

```
// Użycie static_cast i dynamic_cast
#include <iostream>
using namespace std;

class Zwierze
{
public:
	virtual void Glos() = 0;
};

class Pies : public Zwierze
{
public:
	void Aktywnosc(){cout<<"Pies : Macham ogonem
!!"<<endl;}
	void Glos(){cout<<"Pies : Hau hau !!"<<endl;}
};
```

```
class Kot : public Zwierze
{
public:
	void Aktywnosc(){cout<<"Kot : Zlapalem mysz
!!"<<endl;}
	void Glos(){cout<<"Kot : Miau miau !!"<<endl;}
};

void ID(Zwierze* pZwierze);

void main()
{
	Zwierze* pZwierze1 = new Pies;
	Zwierze* pZwierze2 = new Kot;

	ID(pZwierze1);
	pZwierze1->Glos();
	//pZwierze1->Aktywnosc();
	ID(pZwierze2);
	pZwierze2->Glos();
	//pZwierze2->Aktywnosc();
	system("pause");
}
```

```
void ID(Zwierze* pZwierze)
{
	Pies* pPies = static_cast<Pies*>(pZwierze);
	if(pPies)
	{
		cout<<"Zwierze to pies"<<endl;
		pPies->Glos();
		pPies->Aktywnosc();
	}
	Kot* pKot = dynamic_cast<Kot*>(pZwierze);
	if(pKot)
	{cout<<"Zwierze to kot"<<endl;
	pKot->Glos();
	pKot->Aktywnosc();
	}
}
```

```
Zwierze to pies
Pies : Hau hau !!
Pies : Macham ogonem !!
Pies : Hau hau !!
Zwierze to pies
Kot : Miau miau !!
Pies : Macham ogonem !!
Zwierze to kot
Kot : Miau miau !!
Kot : Zlapalem mysz !!
Kot : Miau miau !!
Press any key to continue . . .
```

```
// Użycie reinterpret_cast static_cast dynamic_cast

#include <iostream>
using namespace std;

class Zwierze
{
public:
	virtual void Aktywnosc() = 0;
};

class Pies : public Zwierze
{
public:
	void Aktywnosc(){cout<<"Pies : Macham ogonem
!!"<<endl;}
};

class Kot : public Zwierze
{
public:
	void Aktywnosc(){cout<<"Kot : Zlapalem mysz !!"<<endl;}
};
```

```
void main()
{
        Zwierze* Podworko[8];
        Podworko[0] = new Pies;
        Podworko[1] = new Kot;
        cout<<"Elementy pierwotne :\n";
        Podworko[0]->Aktywnosc();
        Podworko[1]->Aktywnosc();
        Podworko[2]=reinterpret_cast<Pies*>(Podworko[1]);
        Podworko[3]=reinterpret_cast<Kot*>(Podworko[0]);
        cout<<"reinterpret_cast :\nBylem kotem\n";
        Podworko[2]->Aktywnosc();
        cout<<"Bylem psem\n";
        Podworko[3]->Aktywnosc();
        Podworko[4]=static_cast<Pies*>(Podworko[1]);
        Podworko[5]=static_cast<Kot*>(Podworko[0]);
        cout<<"static_cast :\nBylem kotem\n";
        Podworko[4]->Aktywnosc();
        cout<<"Bylem psem\n";
        Podworko[5]->Aktywnosc();
```

```
        /*Podworko[6]=dynamic_cast<Pies*>(Podworko[1]);
        Podworko[7]=dynamic_cast<Kot*>(Podworko[0]);
        cout<<"dynamic_cast :\nBylem kotem\n";
        Podworko[6]->Aktywnosc();
        cout<<"Bylem psem\n";
        Podworko[7]->Aktywnosc();
        program się kompiluje, ale pada przy wykonaniu*/

        system("pause");
}
```

```
Elementy pierwotne :
Pies : Macham ogonem !!
Kot : Zlapalem mysz !!
reinterpret_cast :
Bylem kotem
Kot : Zlapalem mysz !!
Bylem psem
Pies : Macham ogonem !!
static_cast :
Bylem kotem
Kot : Zlapalem mysz !!
Bylem psem
Pies : Macham ogonem !!
Press any key to continue . . .
```

```
// Użycie reinterpret_cast static_cast i dynamic_cast

#include <iostream>
using namespace std;

class Pies
{
public:
	void Aktywnosc(){cout<<"Pies : Macham ogonem !!"<<endl;}
};

class Kot
{
public:
	void Aktywnosc(){cout<<"Kot : Zlapalem mysz !!"<<endl;}
};

void main()
{
	Pies* psy[4];
	Kot* koty[4];
```

```
psy[0] = new Pies;
        koty[0] = new Kot;
        cout<<"Elementy pierwotne :\n";
        psy[0]->Aktywnosc();
        koty[0]->Aktywnosc();
        psy[1]=reinterpret_cast<Pies*>(koty[0]);
        koty[1]=reinterpret_cast<Kot*>(psy[0]);
        cout<<"reinterpret_cast :\nBylem kotem\n";
        psy[1]->Aktywnosc();
        cout<<"Bylem psem\n";
        koty[1]->Aktywnosc();
        /*psy[2]=static_cast<Pies*>(koty[0]);
        koty[2]=static_cast<Kot*>(psy[0]);
        cout<<"static_cast :\nBylem kotem\n";
        psy[2]->Aktywnosc();
        cout<<"Bylem psem\n";
        koty[2]->Aktywnosc();
        psy[3]=dynamic_cast<Pies*>(koty[0]);
        koty[3]=dynamic_cast<Kot*>(psy[0]);
        cout<<"dynamic_cast :\nBylem kotem\n";
        psy[3]->Aktywnosc();
        cout<<"Bylem psem\n";
```

```
        koty[3]->Aktywnosc();
        Pies Azor=reinterpret_cast<Pies>(*koty[0]);*/
        Pies& Reks=reinterpret_cast<Pies&>(*koty[0]);
        cout<<"reinterpret_cast referencja do obiektu
:\nBylem kotem\n";
        Reks.Aktywnosc();

        system("pause");
}
```

```
Elementy pierwotne :
Pies : Macham ogonem !!
Kot : Zlapalem mysz !!
reinterpret_cast :
Bylem kotem
Pies : Macham ogonem !!
Bylem psem
Kot : Zlapalem mysz !!
reinterpret_cast referencja do
obiektu :
Bylem kotem
Pies : Macham ogonem !!
Press any key to continue . . .
```

```
1>e:\praca\dydaktyka\programowanie obiektowe\visual project\wykład
2016\w_10_09\w_10_09\w_10_09.cpp(33) : error C2440: 'static_cast' : cannot
convert from 'Kot *' to 'Pies *'
1>       Types pointed to are unrelated; conversion requires reinterpret_cast,
C-style cast or function-style cast
1>e:\praca\dydaktyka\programowanie obiektowe\visual project\wykład
2016\w_10_09\w_10_09\w_10_09.cpp(34) : error C2440: 'static_cast' : cannot
convert from 'Pies *' to 'Kot *'
1>       Types pointed to are unrelated; conversion requires reinterpret_cast,
C-style cast or function-style cast
1>e:\praca\dydaktyka\programowanie obiektowe\visual project\wykład
2016\w_10_09\w_10_09\w_10_09.cpp(39) : error C2683: 'dynamic_cast' : 'Kot' is
not a polymorphic type
1>       e:\praca\dydaktyka\programowanie obiektowe\visual project\wykład
2016\w_10_09\w_10_09\w_10_09.cpp(13) : see declaration of 'Kot'
1>e:\praca\dydaktyka\programowanie obiektowe\visual project\wykład
2016\w_10_09\w_10_09\w_10_09.cpp(40) : error C2683: 'dynamic_cast' : 'Pies' is
not a polymorphic type
1>       e:\praca\dydaktyka\programowanie obiektowe\visual project\wykład
2016\w_10_09\w_10_09\w_10_09.cpp(7) : see declaration of 'Pies'
1>e:\praca\dydaktyka\programowanie obiektowe\visual project\wykład
2016\w_10_09\w_10_09\w_10_09.cpp(46) : error C2440: 'reinterpret_cast' : cannot
convert from 'Kot' to 'Pies'
1>       Conversion requires a constructor or user-defined-conversion operator,
which can't be used by const_cast or reinterpret_cast
1>Build log was saved at "file://e:\Praca\Dydaktyka\Programowanie
obiektowe\Visual Project\Wykład 2016\W_10_09\W_10_09\Debug\BuildLog.htm"
1>W_10_09 - 5 error(s), 0 warning(s)
```

Standardowa biblioteka wzorców (STL)

Zestaw klas i funkcji wzorcowych dostarczających:

1. Kontenerów do przechowywania informacji
2. Iteratorów służących do uzyskiwania dostępu do przechowywanych informacji
3. Algorytmów pozwalających na manipulowanie treścią znajdującą się w kontenerach

Kontenery STL:

- sekwencyjne (przechowują dane sekwencyjnie – podobnie jak tablice czy listy – krótki czas wstawiania, długi wyszukiwania)
- asocjacyjne (dane w postaci posortowanej – dłuższy czas wstawiania, krótszy wyszukiwania)

Kontenery sekwencyjne

std::vector	Szybkie wstawianie elementu na końcu, dostęp podobny do tablicy	Zmiana wielkości może skutkować spadkiem wydajności, czas wyszukiwania proporcjonalny do liczby elementów
std::deque	Jak vector, możliwość wstawiania elementów na początku	Nie wymaga funkcji reserve () – rezerwacja pamięci
std::list	Stały czas wstawiania na końcu i w środku listy, czas usuwania niezależny od położenia elementu	Brak swobodnego dostępu do do elementów na podstawie ich indeksu

Kontenery asocjacyjne:

std::set, std::multiset,

std::map, std::multimap (pary klucz-wartość)

Podstawowe algorytmy STL:

std::find

std::find_if – wyszukanie wartości

std::reverse – odwrócenie kolejności

std::remove_if – usunięcie

std::transform - przekształcenie

Klasa STL string

Służy do pracy z ciągami tekstowymi. Ułatwia:

- Kopiowanie
- Łączenie ciągów
- Wyszukiwanie znaków i podciągów tekstowych
- Skracanie
- Odwracanie zawartości ciągu tekstowego

```
// Ustanawianie i kopiowanie obiektów klasy string

#include<string>
#include<iostream>
using namespace std;
int main()
{
	const string lancuch_1("ABCDEFGHIJKLMNOP");
	char* lancuch_2 ="ABCDEFGHIJKLMNOP" ;
	//string lancuch_3=const_cast<string>(lancuch_1);
	string lancuch_3=lancuch_2;
	lancuch_3+="QRST";
	cout<<"lancuch_3 : "<<lancuch_3<<endl;
	string kopia_1 (lancuch_1);
	string kopia_2 (lancuch_2);
	cout<<"kopia 1 : "<<kopia_1<<endl;
	cout<<"kopia 2 : "<<kopia_2<<endl;

	string kopia_3(lancuch_3,6);
	cout <<"kopia 3 : " << kopia_3 <<endl;

	string kopia_3_1(lancuch_1,0,6);
	cout <<"kopia 3_1 : " << kopia_3 <<endl;
```

```
        string kopia_4(lancuch_1,4,6);
        cout <<"kopia 4 : " << kopia_4 <<endl;

        string kopia_5(lancuch_2,6);
        cout <<"kopia 5 : " << kopia_5 <<endl;

        string kopia_6(lancuch_1,4,6);
        cout <<"kopia 6 : " << kopia_6 <<endl;

        //Inicjalizacja obiektu z 4 znakami X
        string lancuch_4(4, 'X');
        cout <<"lancuch_4 :" << lancuch_4 <<endl;
        system("pause");
}
```

```
lancuch_3 : ABCDEFGHIJKLMNOPQRST
kopia 1 : ABCDEFGHIJKLMNOP
kopia 2 : ABCDEFGHIJKLMNOP
kopia 3 : GHIJKLMNOPQRST
kopia 3_1 : GHIJKLMNOPQRST
kopia 4 : EFGHIJ
kopia 5 : ABCDEF
kopia 6 : EFGHIJ
lancuch_4 :XXXX
Press any key to continue . . .
```

```
// Dostęp do elementów składowych obiektu STL string

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

void main()
{
	string lancuch_1("abcdefghijkl");

	cout <<"Wyswietlenie zawartosci za pomoca tablicy :"
<<endl;
	for (int k = 0;	k < lancuch_1.length();	++ k)
	{
		cout <<"Znak [" << k <<"] to: ";
		cout << lancuch_1[k] <<endl;
	}
	cout <<endl;
```

```
//Dostęp do ciągu za pomocą iteratorów
	cout <<"Wyswietlenie znakow za pomoca iteratorow :"
<<endl;
	int i = 0;
	string::iterator iter;
	for (iter = lancuch_1.begin()
		; iter != lancuch_1.end();
		++ iter)
	{
		cout <<"Znak [" << i ++ <<"] to: ";
		cout << *iter <<endl;
	}
	cout <<endl;
	// Dostęp do zawartości jak do ciągu w stylu C

	cout << "Reprezentacja char* ciagu to : ";
	cout << lancuch_1.c_str() <<endl;
	system("pause");

}
```

```
Wyswietlenie zawartosci za
pomoca tablicy :
Znak [0] to: a
Znak [1] to: b
Znak [2] to: c
Znak [3] to: d
Znak [4] to: e
Znak [5] to: f
Znak [6] to: g
Znak [7] to: h
Znak [8] to: i
Znak [9] to: j
Znak [10] to: k
Znak [11] to: l
```

```
Wyswietlenie znakow za pomoca
iteratorow :
Znak [0] to: a
Znak [1] to: b
Znak [2] to: c
Znak [3] to: d
Znak [4] to: e
Znak [5] to: f
Znak [6] to: g
Znak [7] to: h
Znak [8] to: i
Znak [9] to: j
Znak [10] to: k
Znak [11] to: l

Reprezentacja char* ciagu to :
abcdefghijkl
Press any key to continue . . .
```

```
// Łączenie ciągów tekstowych

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

void main()
{
	string przyklad_1 ("Przyklad 1");
	string przyklad_2 (" i drugi.");
	//Łączenie
	przyklad_1 += przyklad_2;
	cout << przyklad_1 << endl;

	string przyklad_3 (" Mozna dodac trzeci !");
	przyklad_1.append (przyklad_3);
	cout << przyklad_1 << endl;
	char* wskaznik = " Wskaznik tez mozna dodac !";
	przyklad_1.append(wskaznik);
	cout << przyklad_1 << endl;
	system("pause");
}
```

```
Przyklad 1 i drugi.
Przyklad 1 i drugi. Mozna dodac trzeci !
Przyklad 1 i drugi. Mozna dodac trzeci ! Wskaznik tez mozna
dodac !
Press any key to continue . . .
```