```
// Wskaźniki

#include<iostream>
using namespace std;

class Owad
{
public:
	int waga;
	bool jadowitosc;

	Owad():waga(1),jadowitosc(false)
	{cout<<"Konstruktor domyslny owada\n";}

	Owad (bool j):waga(2),jadowitosc(j)
	{cout<<"Konstruktor owada\n";}

	~Owad()
	{cout<<"Destruktor owada\n";}

	void Ruch()
	{cout<<"Owad lata\n";}
};
```

```
class Komar : public Owad
{
public:
	bool brudas;

	Komar(): Owad(true),brudas(false)
	{cout<<"Konstruktor komara\n";}

	void Aktywnosc()
	{cout<<"Brzeczy i gryzie\n";}

	~Komar ()
	{cout<<"Komar rozgnieciony na scianie\n";}
};


class Mucha : public Owad
{
public:
	bool brudas;

	Mucha(): brudas(true)
	{cout<<"Konstruktor muchy\n";}
```

```
void Aktywnosc()
	{cout<<"Lata i brzeczy\n";}

	~Mucha ()
	{cout<<"Mucha dostala w beret\n";}
};

int main()
{
	Owad* owad=new Owad;
	cout<<"Rozmiar obiektu : ";
	cout<<sizeof(owad);
	cout<<"\nOwad :\n";
	cout<<"waga - "<<owad->waga;
	cout<<"\njadowitosc - "<<owad->jadowitosc<<endl;
	owad->Ruch();
	delete owad;
	cout<<endl;

	Owad* mucha=new Mucha;
	cout<<"Rozmiar obiektu : ";
	cout<<sizeof(mucha);
	cout<<"\nMucha :\n";
```

```
cout<<"waga - "<<mucha->waga;
cout<<"\njadowitosc - "<<mucha->jadowitosc<<endl;
//cout<<"\nbrudas - "<<mucha->brudas;
mucha->Ruch();
//mucha->Aktywnosc();
delete mucha;
cout<<endl;

Owad* komar=new Komar;
cout<<"Rozmiar obiektu : ";
cout<<sizeof(komar);
cout<<"\nKomar :\n";
cout<<"waga - "<<komar->waga;
cout<<"\njadowitosc - "<<komar->jadowitosc<<endl;
//cout<<"\nbrudas - "<<komar->brudas;
komar->Ruch();
//komar->Aktywnosc();
delete komar;
cout<<endl;
```

```
Mucha* mucha_1=new Mucha;
/*pomimo delete mucha jest pamiętana jako wskaźnik do
klasy Owad*/
cout<<"Rozmiar obiektu : ";
cout<<sizeof(mucha_1);
cout<<"\nMucha :\n";
cout<<"waga - "<<mucha_1->waga;
cout<<"\njadowitosc - "<<mucha_1->jadowitosc;
cout<<"\nbrudas - "<<mucha_1->brudas<<endl;
mucha_1->Ruch();
mucha_1->Aktywnosc();
delete mucha_1;
cout<<endl;

Komar* komar_1=new Komar;
//j. w.
cout<<"Rozmiar obiektu : ";
cout<<sizeof(komar_1);
cout<<"\nKomar :\n";
cout<<"waga - "<<komar_1->waga;
cout<<"\njadowitosc - "<<komar_1->jadowitosc;
cout<<"\nbrudas - "<<komar_1->brudas<<endl;
```

```
komar_1->Ruch();
        komar_1->Aktywnosc();
        delete komar_1;
        cout<<endl;

        /*
        Mucha* mucha_2=new Owad;
        wymagane rzutowanie - o tym później
        */

        system("pause");
        return 0;
}
```

```
Konstruktor domyslny owada
Rozmiar obiektu : 4
Owad :
waga - 1
jadowitosc - 0
Owad lata
Destruktor owada

Konstruktor domyslny owada
Konstruktor muchy
Rozmiar obiektu : 4
Mucha :
waga - 1
jadowitosc - 0
Owad lata
Destruktor owada

Konstruktor owada
Konstruktor komara
Rozmiar obiektu : 4
Komar :
waga - 2
```

```
jadowitosc - 1
Owad lata
Destruktor owada

Konstruktor domyslny owada
Konstruktor muchy
Rozmiar obiektu : 4
Mucha :
waga - 1
jadowitosc - 0
brudas - 1
Owad lata
Lata i brzeczy
Mucha dostala w beret
Destruktor owada
```

```
Konstruktor owada
Konstruktor komara
Rozmiar obiektu : 4
Komar :
waga - 2
jadowitosc - 1
brudas - 0
Owad lata
Brzeczy i gryzie
Komar rozgnieciony na scianie
Destruktor owada

Press any key to continue . . .
```

```
// Rzutowanie

#include<iostream>
using namespace std;

class Owad
{
public:
	int waga;
	bool jadowitosc;

	Owad():waga(1),jadowitosc(false)
	{cout<<"Konstruktor domyslny owada\n";}

	~Owad()
	{cout<<"Destruktor owada\n";}

	void Ruch()
	{cout<<"Owad lata\n";}
};
```

```
class Komar : public Owad
{
public:
	bool brudas;

	Komar(): Owad() ,brudas(false)
	{cout<<"Konstruktor komara\n";}

	void Aktywnosc()
	{cout<<"Brzeczy i gryzie\n";}

	~Komar ()
	{cout<<"Komar rozgnieciony na scianie\n";}
};


int main()
{
	cout<<"static_cast\n\n";
	Komar* komar=static_cast<Komar*>(new Owad);
	cout<<"waga : "<<komar->waga<<endl;
	cout<<"jadowitosc : "<<komar->jadowitosc<<endl;
```

```
cout<<"brudas : "<<komar->brudas<<endl;
	komar->Aktywnosc();
	komar->Ruch();
	delete komar;

	//Komar* komar_1=dynamic_cast<Komar*>(new Owad);
	cout<<"reinterpret_cast\n\n";
	Komar* komar_1=reinterpret_cast<Komar*>(new Owad);
	cout<<"waga : "<<komar_1->waga<<endl;
	cout<<"jadowitosc : "<<komar_1->jadowitosc<<endl;
	cout<<"brudas : "<<komar_1->brudas<<endl;
	komar_1->Aktywnosc();
	komar_1->Ruch();
	delete komar_1;

	system("pause");
	return 0;
}
```

```
static_cast

Konstruktor domyslny owada
waga : 1
jadowitosc : 0
brudas : 253
Brzeczy i gryzie
Owad lata
Komar rozgnieciony na scianie
Destruktor owada
reinterpret_cast

Konstruktor domyslny owada
waga : 1
jadowitosc : 0
brudas : 253
Brzeczy i gryzie
Owad lata
Komar rozgnieciony na scianie
Destruktor owada
Press any key to continue . . .
```

Przesłanianie funkcji klasy bazowej w klasie potomnej

Zmiana implementacji funkcji klasy bazowej w klasie pochodnej – zachodzi, gdy klasa pochodna tworzy funkcję o tym samym zwracanym typie, ale z inna implementacją – funkcja klasy bazowej została przesłonięta (override)

```
// Przeslanianie funkcji klasy
// bazowej w klasie potomnej

#include<iostream>
using std::cout;
using std::endl;

class Ssak
{public:

void Glos()
{cout<<"Glos ssaka\n";}
};

class Kot : public Ssak
{
public:
       void Glos()
       {cout<<"Miau !!\n";}
};
```

```
void main()
{
	Ssak zwierz;
	zwierz.Glos();
	Kot Mruczek;
	Mruczek.Glos();
	Mruczek.Ssak::Glos();
	system("pause");
}
```

```
 Glos ssaka
 Miau !!
 Glos ssaka
 Press any key to continue . . .
```

```
// Przeslanianie funkcji klasy
// bazowej w klasie potomnej

#include<iostream>
using std::cout;
using std::endl;

class Ssak
{public:

void Ruch(int m)
{cout<<"Ssak przeszedl "<<m<<" krokow\n";}
};

class Kot : public Ssak
{
public:
       void Ruch()
       {cout<<"Kot przeszedl 10 krokow\n";}
};
```

```
void main()
{
	Ssak zwierz;
	zwierz.Ruch(5);
	Kot Mruczek;
	Mruczek.Ruch();
	//Mruczek.Ruch(6);
	Mruczek.Ssak::Ruch(8);
	system("pause");
}
```

```
Ssak przeszedl 5 krokow
Kot przeszedl 10 krokow
Ssak przeszedl 8 krokow
Press any key to continue . . .
```

```
//Działanie modyfikatora const

#include <iostream>
using namespace std;

class Czolg
{
	public:
		void Armata()
		{cout<<"armata"<<endl;}
		void Naped()
		{cout<<"naped"<<endl;}
		void Zaloga()const
		{cout<<"4 - 5"<<endl;}

};
class T34 : public Czolg
{
public:
		void Armata()
		{cout<<"F-34"<<endl;}
```

```
void Naped() const
             {cout<<"diesel"<<endl;}
             void Zaloga()
             {cout<<"4"<<endl;}
};
class Panther: public Czolg
{
       public:
             void Armata()const
             {cout<<"KwK-43"<<endl;}
             void Naped()
             {cout<<"benzyna"<<endl;}
             void Zaloga()const
             {cout<<"5"<<endl;}
};

void main()
{
       cout<<"Tworzymy obiekty :\n";
       T34 Rudy;
       cout<<"\nT - 34\n";
       Rudy.Armata();
       Rudy.Naped();
```

```
Rudy.Zaloga();
	Panther Nr_101;
	cout<<"\nPanther\n";
	Nr_101.Armata();
	Nr_101.Naped();
	Nr_101.Zaloga();
	cout<<"\n Wskazniki do klasy czolg\n";
	Czolg* czolgi[4];
	cout<<"\nT - 34\n";
	czolgi[0]=new T34;
	czolgi[0]->Armata();
	czolgi[0]->Naped();
	czolgi[0]->Zaloga();
	czolgi[1]=new Panther;
	cout<<"\nPanther\n";
	czolgi[1]->Armata();
	czolgi[1]->Naped();
	czolgi[1]->Zaloga();

	system ("pause");
}
```

```
Tworzymy obiekty :

T - 34
F-34
diesel
4

Panther
KwK-43
benzyna
5

 Wskazniki do klasy czolg

T - 34
armata
naped
4 - 5

Panther
armata
naped
4 - 5
Press any key to continue . . .
```

Funkcje (metody) wirtualne

Obiekt klasy Pies jest jednocześnie obiektem klasy Ssak. W C++ możliwe jest przypisywanie wskaźnikom do klas bazowych wskaźników do klas pochodnych, np.

Ssak* pSsak = new Pies;

Na stercie tworzymy nowy obiekt klasy Pies, zaś otrzymany wskaźnik przypisujemy wskaźnikowi do obiektów klasy Ssak. Wskaźnik ten może zostać wykorzystany do wywołania dowolnej metody klasy Ssak

```
// Metody wirtualne

#include<iostream>
using namespace std;

class Ssak
{
public:

	void Ruch() {cout<<"Ssak chodzi\n";}
	void Sen() {cout<<"Ssak spi\n";}
	virtual void Glos(){cout<<"Glos ssaka\n";}
	virtual void Glod() {cout<<"Ssak je\n";}
};
class Pies : public Ssak
{
public:
	void Glos(){cout<<"Hau\n";}
	void Ruch(){cout<<"Pies biega\n";}
	virtual void Sen() {cout<<"Pies spi\n";}
	virtual void Glod() {cout<<"Pies je\n";}
};
```

```
void main()
{
	Ssak *pPies = new Pies;
	pPies->Glod();
	pPies->Glos();
	pPies->Ruch();
	pPies->Sen();
	system("pause");
}
```

```
 Pies je
 Hau
 Ssak chodzi
 Ssak spi
 Press any key to continue . . .
```

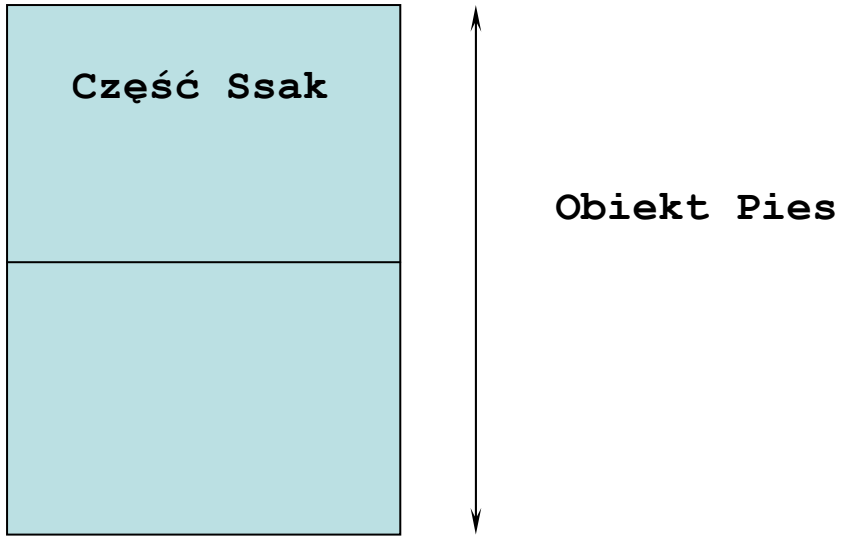
Funkcja	Ssak	Pies	Przesłonięcie
`Ruch()`			NIE
`Sen()`		`virtual`	NIE
`Glos()`	`virtual`		TAK
`Glod()`	`virtual`	`virtual`	TAK

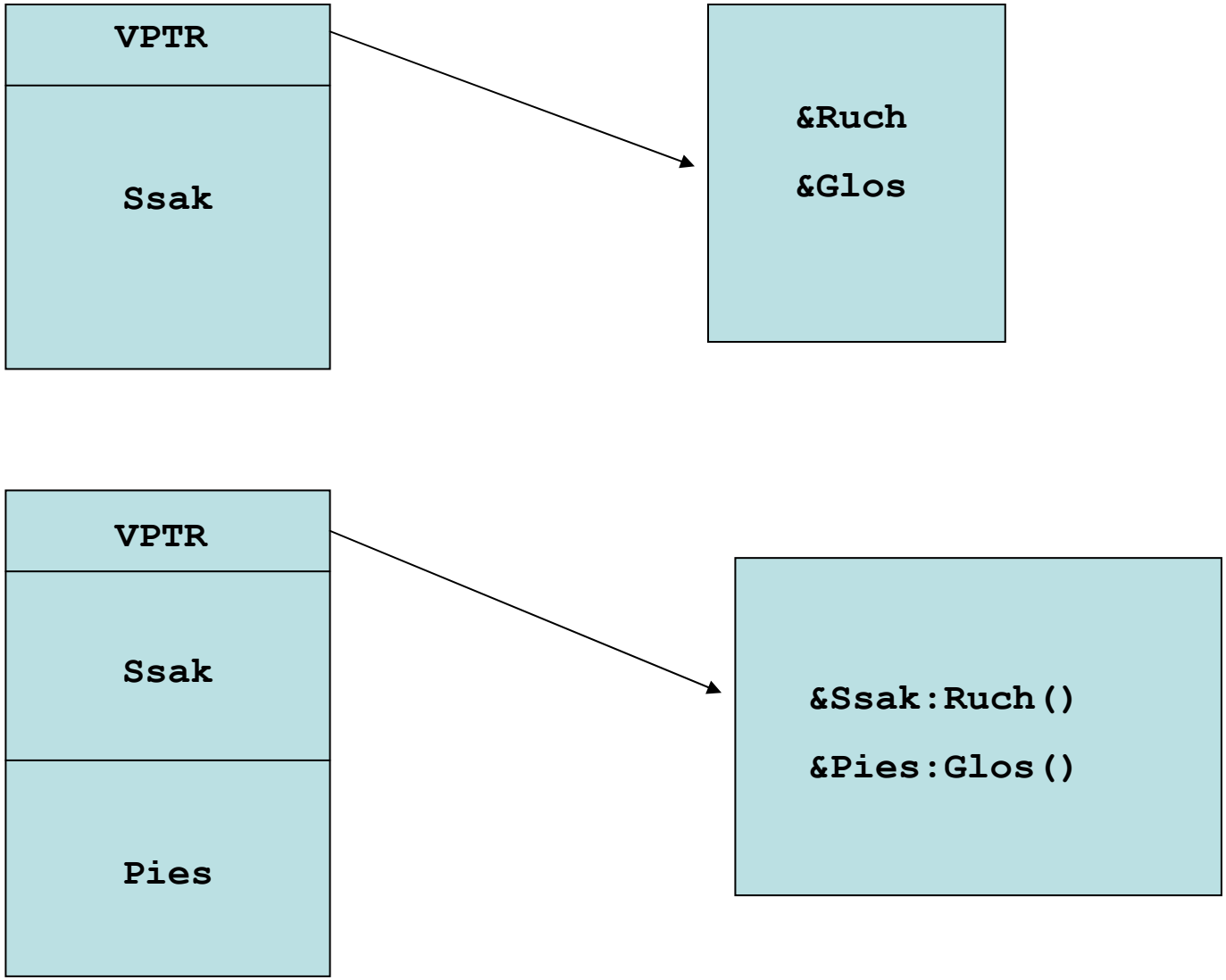
Wniosek:

Funkcje wirtualne w klasie bazowej są przesłonięte

Działanie funkcji wirtualnych

Funkcja wirtualna jest śledzona przez obiekt – kompilator tworzy tablicę funkcji wirtualnych v-table, która jest przechowywana dla każdego typu, a każdy obiekt tego typu przechowuje wskaźnik do niej vptr, vpointer.




VPTR
Ssak
&Ruch
&Glos
VPTR
Ssak
Pies
&Ssak:Ruch()
&Pies:Glos()

```
// Wielokrotne użycie funkcji wirtualnych

#include<iostream>
using std::cout;
using std::cin;

class Ssak
{
public:

	virtual void Glos(){cout<<"Glos ssaka\n";}
};
class Pies : public Ssak
{
public:
	void Glos(){cout<<"Hau\n";}
};

class Kot : public Ssak
{
public:
	void Glos(){cout<<"Miau\n";}
};
```

```
class Kon : public Ssak
{
public:
	void Glos(){cout<<"Ihaaaa!\n";}
};

class Swinka : public Ssak
{
public:
	void Glos(){cout<<"Kwik!\n";}
};
void main()
{
	Ssak* zwierzeta[5];
	Ssak* ptr;
	int wybor, i;
	for (i = 0; i < 5; i++)
	{
		cout<<"(1)pies	(2)kot (3)kon (4)swinka : ";
		cin>>wybor;
```

```
switch (wybor)
                {
                case 1: ptr = new Pies;
                        break;
                case 2: ptr = new Kot;
                        break;
                case 3: ptr = new Kon;
                        break;
                case 4: ptr = new Swinka;
                        break;
                default: ptr = new Ssak;
                        break;
                }
        zwierzeta[i] = ptr;
        }

        for (i = 0; i < 5; i++)
                zwierzeta[i]->Glos();

        system("pause");
}
```

```
(1)pies (2)kot  (3)kon  (4)swinka : 0
(1)pies (2)kot  (3)kon  (4)swinka : 1
(1)pies (2)kot  (3)kon  (4)swinka : 2
(1)pies (2)kot  (3)kon  (4)swinka : 3
(1)pies (2)kot  (3)kon  (4)swinka : 4
Glos ssaka
Hau
Miau
Ihaaaa!
Kwik!
Press any key to continue . . .
```

```
/* Okrajanie przy przekazywaniu
argumentów przez wartość*/

#include<iostream>
using namespace std;

class Ssak
{
public:
	virtual void Glos(){cout<<"Glos ssaka\n";}
};
class Pies : public Ssak
{
public:
	void Glos(){cout<<"Hau\n";}
};

class Kot : public Ssak
{
public:
	void Glos(){cout<<"Miau\n";}
};
```

```
/*przekazanie parametrów
przez wartość, wskaźnik i referencję*/

void Wartosc(Ssak s)
{s.Glos();}
void Wskaznik(Ssak* p)
{p->Glos();}
void Referencja(Ssak& r)
{r.Glos(); }

void main()
{
	Ssak* ptr = 0;
	int wybor;
	while (1)
	{
		bool Wyjscie = false;
		cout<<"(1)pies	(2)kot (0)wyjscie : ";
		cin>>wybor;
```

```
switch (wybor)
                {
                case 0: Wyjscie = true;
                        break;
                case 1: ptr = new Pies;
                        break;
                case 2: ptr = new Kot;
                        break;
                default: ptr = new Ssak;
                        break;
                }
                if(Wyjscie)
                        break;
                Wskaznik(ptr);
                Wartosc(*ptr);
                Referencja(*ptr);
        }
        system("pause");
}
```

```
(1)pies (2)kot  (0)wyjscie : 1
Hau
Glos ssaka
Hau
(1)pies (2)kot  (0)wyjscie : 2
Miau
Glos ssaka
Miau
(1)pies (2)kot  (0)wyjscie : 0
Press any key to continue . . .
```

```
//Działanie modyfikatora const
//w funkcjach wirtualnych

#include <iostream>
using namespace std;

class Czolg
{
public:
              virtual void Armata()
              {cout<<"armata"<<endl;}
              virtual void Naped()
              {cout<<"naped"<<endl;}
              virtual void Zaloga()const
              {cout<<"4 - 5"<<endl;}
};
class T34 : public Czolg
{
public:
              virtual void Armata()
              {cout<<"F-34"<<endl;}
```

```
                virtual void Naped() const
                {cout<<"diesel"<<endl;}
                virtual void Zaloga()
                {cout<<"4"<<endl;}
};
class Panther: public Czolg
{
        public:
                virtual void Armata()const
                {cout<<"KwK-43"<<endl;}
                virtual void Naped()
                {cout<<"benzyna"<<endl;}
                virtual void Zaloga()const
                {cout<<"5"<<endl;}
};

void main()
{
        cout<<"Tworzymy obiekty :\n\n";
        T34 Rudy;
        cout<<"\nT - 34\n";
        Rudy.Armata();
```

```
Rudy.Naped();
        Rudy.Zaloga();
        Panther Nr_101;
        cout<<"\nPanther\n";
        Nr_101.Armata();
        Nr_101.Naped();
        Nr_101.Zaloga();
        cout<<"\n Wskazniki do klasy czolg\n";
        Czolg* czolgi[4];
        cout<<"\nT - 34\n";
        czolgi[0]=new T34;
        czolgi[0]->Armata();
        czolgi[0]->Naped();
        czolgi[0]->Zaloga();
        czolgi[1]=new Panther;
        cout<<"\nPanther\n";
        czolgi[1]->Armata();
        czolgi[1]->Naped();
        czolgi[1]->Zaloga();
        system ("pause");
}
```

```
Tworzymy obiekty :

T - 34
F-34
diesel
4

Panther
KwK-43
benzyna
5

 Wskazniki do klasy czolg

T - 34
F-34
naped
4 - 5

Panther
armata
benzyna
5
Press any key to continue . . .
```

Funkcja	Czolg	T34	Panther
Armata()			const
Naped()		const	
Zaloga()	const		const

Wniosek:

Zakreślone są funkcje przesłonięte w klasie potomnej. Aby funkcja została przesłonięta, musi być const lub nie w obu klasach : bazowej i potomnej. Jeśli modyfikator const występuje tylko w jednej z nich – funkcja klasy bazowej nie jest przesłonięta w klasie potomnej

Przeładowanie i przesłanianie (polimorfizm) funkcji	
Przeładowanie funkcji	Przesłanianie funkcji
Funkcje mają te same nazwy, należą do tej samej klasy, różnią się listą parametrów lub zwracanym typem	Funkcje mają te same nazwy, te same listy parametrów i typy zwracane, ale należą do różnych klas