4

Techniki programowania obiektowego (C++)

Przeciążanie funkcji

różna liczba argumentów, różne typy argumentów, parametry domyślne, rozstrzyganie

```
//————————————przeciazanie funkcji (overloading)
#include ...
void fun(int i) {cout<<"\nfun1: i= "<<i;}
void fun(double d) {cout<<"\nfun2: d= "<<d;}
void main(void)
{
clrscr();
 fun(5);
 fun(5.0);
}
```

```
//————————————argumenty opcjonalne, wartosci domyslne
#include ...
int sum(int a, int b, int c=10, int mnoznik=1)
{
 return( mnoznik*(a+b+c) );
}
void main(void)
{
 clrscr();
 cout<<"\n"<< sum(0,0);
 cout<<"\n"<< sum(0,0,1);
 cout<<"\n"<< sum(1,1,100,2);
}
```

Szablony funkcji – parametryzacja typu

```
//——————————————szablon funkcji
#include ...
T gre(T x, T y)
{
 if(x>y) return(x); //funkcja gre() zadziala dla wszystkich
 else return(y); //typow (klas) dla ktorych zdefiniowany jest
} //operator porownania >
void main()
{ clrscr();
 int i=0,j=1; cout<<gre(i,j);
 double x=0.22,y=0.88; cout<<"\n"<<gre(x,y);
}
```

Szablony klas – parametryzacja typu c.d.

```
//———————— szablon klasy
#include ...
template <class T>
class Vector
{
 T *data;
 int size;
public:
 Vector(int);
 ~Vector( ) { delete[ ] data; }
 T& operator[ ] (int i) { return data[i]; } //przeciazany operator []
};
```

```
template <class T> Vector<T>::Vector(int n) { data = new T[n];
 size = n;
};
void main()
{
 Vector<int> x(3); // skonstruowanie wektora 3-elementowego typu int
 x[0] = 1; x[1] = 2; x[2] = 3;
 cout<<"\n"<<x[0]<<"\n"<<x[1]<<"\n"<<x[2];
 getch();
}
```

Przykład listy dynamicznej zrealizowanej obiektowo

```
//———————————kolejka (lista) obiektowa
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
enum bool{false,true};
template <class element> // szablon klasy
class Lista
{
 struct ElemListy //struktura do stworzenia listy
 {
 element wartosc;
 ElemListy *nast;
 } *start;
 public:
 Lista(); //konstruktor
 ~Lista(); //destruktor
```

```
 void zKolejki(); //kasuje 1-szy element z kolejki-f pomocnicza do destr
 bool jestPusta() const; //sprawdzenie czy jest pusty nic nie znienia
 void doKolejki(const element &elem);//dopisuje element na koncu
 element jestPierwszy() const; //odczytuje elem pocz, nic nie zmienia
};
//*****************METODY (FUNKCJE SKADOWE) KLASY Lista*************
template <class element>
Lista<element>::Lista() // konstruktor
{
 start=NULL;
}
template <class element>
Lista<element>::~Lista() // destruktor
{
 ElemListy *pomoc1=start;
 ElemListy *pomoc2;
 while(pomoc1->nast!=NULL)
 {
 pomoc2=pomoc1;
 pomoc1=pomoc1->nast;
 delete pomoc2;
 };
 start=NULL;
}
template <class element>
element Lista<element>::jestPierwszy() const
{
 if (start!=NULL) return start->wartosc; //zwraca wartosc pierwszego elem
 else return 0;
}
template <class element>
```

```
bool Lista<element>::jestPusta() const // sprawdzenie czy jest pusta
{
 if (start==NULL) return true;
 else return false;
}
template <class element>
void Lista<element>::zKolejki()
{
 if (start!=NULL)
 {
 ElemListy *pomoc=start;
 start=start->nast;
 delete pomoc;
 }
}
template <class element>
void Lista<element>::doKolejki(const element &dolaczany)
{
 ElemListy *pomoc;
 ElemListy *nowy = new ElemListy;
 nowy->nast=NULL;
 nowy->wartosc=dolaczany;
 if (start==NULL) start=nowy;
 else
 {
 pomoc=start;
 while (pomoc->nast!=NULL) pomoc=pomoc->nast;
 pomoc->nast=nowy;
 }
}
void main(){..... //——funkcje main pominieto
```

Dziedziczenie

```
class matrix //klasa bazowa
{
 double **p;
 int s1,s2;
public:
 ...
matrix(int i, int j) {...};
double& element(int i, int j) {...};
...
};


class wiersz: public matrix
{public:
 wiersz(int d2); //konstruktor – nie dziedziczy się
 double& element(int j); //jednoargumentowy
};


wiersz::wiersz(int d2):matrix(1,d2)
{}


double& wiersz::element(int j)
{return(matrix::element(1,j));}
```

Standardowa biblioteka szablonów – STL

szablony: Vector, List, Set, Map,...

algorytmy uogólnione: sort(), aggregate(), resize(),...