1. Obiekt, klasa, pola, metody,
2. Tablice.

Przykład 1 (C++) Tworzenie nowych obiektów. Pojęcia związane z tworzeniem nowych obiektów.

[p1.cpp]

```
#include <iostream>
#include <string>
#include <cstdlib>

using namespace std;

class Pies
{
    //pola
    //private - dostep do zmiennych tylko poprzez
    //funkcje skladowe (metody)
    private:
        string imie;
        double waga;

    //metody
    public:
        //konstruktor domyslny
        Pies() : imie("Brak"), waga(5){
            cout << "Konstruktor domysny" << endl;
        }

        //konstruktor z parametrami
        Pies(string i, double w=10){
            imie=i;
            waga=w;
            cout << "Konstruktor z parametrami"<<endl;
        }

        void info(){
            cout<<"imie: "<<imie<<", waga: "<<waga<<"kg"<<endl;
        }
};

int main()
{
    int a;      //a jest obiektem typu int
    a=20;

    Pies x;     //x jest obiektem typu/klasy pies
    x.info();   //wywołanie metody na rzecz obiektu x

    x=Pies("Azor",a);
    x.info();

    Pies y("Burek");
    y.info();

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

Tablica jest fragmentem pamięci o określonej liczbie elementów.
Poniżej poznamy sposób zarządzania tablicami (wstawianie nowych wartości do tablicy, kasowanie elementu zawierającego określoną wartość, szukanie elementu o określonej wartości).

Przykład 2 (C++) Przykład programu proceduralnego na użycie tablic (array).

[array.cpp]

```
//array.cpp
//demonstrates arrays
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
int main()
{
    int arr[100];               //array
    int nElems = 0;             //number of items
    int j;                      //loop counter
    int searchKey;              //key of item to search for
//-----------------------------------------------------------
    arr[0] = 77;                //insert 10 items
    arr[1] = 99;
    arr[2] = 44;
    arr[3] = 55;
    arr[4] = 22;
    arr[5] = 88;
    arr[6] = 11;
    arr[7] = 00;
    arr[8] = 66;
    arr[9] = 33;
    nElems = 10;                //now 10 items in array
//-----------------------------------------------------------
    for(j=0; j<nElems; j++)     //display items
        cout << arr[j] << " ";
    cout << endl;
//-----------------------------------------------------------
    searchKey = 66;             //find item with key 66
    for(j=0; j<nElems; j++)     //for each element,
        if(arr[j] == searchKey) //found item?
            break;              //yes, exit before end
    if(j == nElems)             //at the end?
        cout << "Can't find " << searchKey << endl; //yes
    else
        cout << "Found " << searchKey << endl;      //no
//-----------------------------------------------------------
    searchKey = 55;             //delete item with key 55
    cout << "Deleting " << searchKey << endl;
    for(j=0; j<nElems; j++)     //look for it
        if(arr[j] == searchKey)
            break;
    for(int k=j; k<nElems; k++) //move higher ones down
        arr[k] = arr[k+1];
    nElems--;                   //decrement size
//-----------------------------------------------------------
    for(j=0; j<nElems; j++)     //display items
```

```
        cout << arr[j] << " ";
    cout << endl;
    return 0;
}  //end main()
```

Wstawianie nowych elementów.
Wstawianie do tablicy jest proste:
arr[0] = 77;

Wyszukiwanie elementów.
Zmienna searchKey przechowuje poszukiwaną wartość. W celu znalezienia elementu w tablicy o wartości takiej jak searchKey przeglądamy cała tablicę krok po kroku. W momencie kiedy zmienna j osiągnie ostatnią zajętą komórkę bez trafienia oznacza to, że szukanej wartości nie ma w tablicy.

Kasowanie elementów.
Kasowanie zaczyna się od znalezienia określonego elementu. Jeśli element o określonej, szukanej wartości występuje w tablicy to po jego znalezieniu przenosimy wszystkie elementy o indeksie większym niż kasowany w dół o jeden element. Następnie zmniejszamy wartość zmiennej nElems.

Wyświetlanie zawartości tablicy.
Przebiegamy całą tablicę wyświetlając wartość każdego elementu tablicy.

Stopniowo w dwóch następnych przykładach do zagadnienia wykorzystania tablic, będziemy podchodzić w sposób obiektowo zorientowany.

Przykład 3 (C++) Podejście pierwsze.

[lowArray.cpp]

```
//lowArray.cpp
//demonstrates array class with low-level interface
#include <iostream>
#include <vector>
using namespace std;
////////////////////////////////////////////////////////////////
class LowArray
{
    private:
        vector<double> v;                   //vector holds doubles

    public:
//--------------------------------------------------------------
    LowArray(int max)                       //constructor
    { v.resize(max); }                      //size the vector
//--------------------------------------------------------------
    void setElem(int index, double value)   //put element into
    { v[index] = value; }                   //array, at index
//--------------------------------------------------------------
    double getElem(int index)               //get element from
    { return v[index]; }                    //array, at index
    //--------------------------------------------------------------
```

```
};  //end class LowArray
////////////////////////////////////////////////////////////////
int main()
{
    LowArray arr(100);                      //create a LowArray
    int nElems = 0;                         //number of items
    int j;                                  //loop variable
//-----------------------------------------------------------
    arr.setElem(0, 77);                     //insert 10 items
    arr.setElem(1, 99);
    arr.setElem(2, 44);
    arr.setElem(3, 55);
    arr.setElem(4, 22);
    arr.setElem(5, 88);
    arr.setElem(6, 11);
    arr.setElem(7, 00);
    arr.setElem(8, 66);
    arr.setElem(9, 33);
    nElems = 10;                            //now 10 items in array
//-----------------------------------------------------------
    for(j=0; j<nElems; j++)                 //display items
        cout << arr.getElem(j) << " ";
    cout << endl;
//-----------------------------------------------------------
    int searchKey = 26;                     //search for item
    for(j=0; j<nElems; j++)                 //for each element,
        if(arr.getElem(j) == searchKey)         //found item?
            break;
    if(j == nElems)                         //no
        cout << "Can't find " << searchKey << endl;
    else                                    //yes
        cout << "Found " << searchKey << endl;
//-----------------------------------------------------------
    double deleteKey = 55;                  //delete value 55
    cout << "Deleting element " << deleteKey << endl;
    for(j=0; j<nElems; j++)                 //look for it
        if(arr.getElem(j) == deleteKey)
            break;
    for(int k=j; k<nElems; k++)             //higher ones down
        arr.setElem(k, arr.getElem(k+1) );
    nElems--;                               //decrement size
//-----------------------------------------------------------
    for(j=0; j<nElems; j++)                 //display items
        cout << arr.getElem(j) << " ";
    cout << endl;
    return 0;
}  //end main()
```

Zamiast tablic użyliśmy klasy vector. Pod wieloma względami operacje na obiektach typu vector są podobne do operacji na tablicach. Użycie kasy vector pozwala nam na określenie rozmiaru wektora (ma to miejsce w konstruktorze klasy lowArray przy wykorzystaniu metody resize()). Pozwala to na przechowywanie dowolnej liczby danych w przeciwieństwie do tablic gdzie ograniczeni byliśmy do tej samej wielkości tablicy.

W dalszej części często strukturę przechowującą dane (vector) nazywać będziemy po prostu tablicą.

W klasie lowArray tablica jest ukryta przed światem zewnętrznym. Tylko funkcje składowe

klasy lowArray mogą uzyskać dostęp do tablicy (private). Tymi funkcjami są: setElem(), getElem() oraz konstruktor.

Przykład 4 (C++) Podejście drugie.

[highArray.cpp]

```
//highArray.cpp
//demonstrates array class with high-level interface
#include <iostream>
#include <vector>
using namespace std;
////////////////////////////////////////////////////////////////
class HighArray
{
    private:
        vector<double> v;                 //vector v
        int nElems;                       //number of data items
    public:
//-----------------------------------------------------------
    HighArray() : nElems(0)               //default constructor
        {  }
//-----------------------------------------------------------
    HighArray(int max) : nElems(0)        //1-arg constructor
        { v.resize(max); }                //size the vector
//-----------------------------------------------------------
    bool find(double searchKey)           //find specified value
    {
        int j;
        for(j=0; j<nElems; j++)           //for each element,
            if(v[j] == searchKey)         //found item?
                break;                    //exit loop before end
        if(j == nElems)                   //gone to end?
            return false;                 //yes, can't find it
        else
            return true;                  //no, found it
    }  //end find()
//-----------------------------------------------------------
    void insert(double value)             //put element into array
    {
        v[nElems] = value;                //insert it
        nElems++;                         //increment size
    }
//-----------------------------------------------------------
    bool remove(double value)             //remove element from array
    {
        int j;
        for(j=0; j<nElems; j++)           //look for it
            if( value == v[j] )
                break;
        if(j==nElems)                     //can't find it
            return false;
        else                              //found it
        {
            for(int k=j; k<nElems; k++)   //move higher ones down
                v[k] = v[k+1];
            nElems--;                     //decrement size
            return true;
        }
```

```
    }  //end delete()
//--------------------------------------------------------------
    void display()                      //displays array contents
    {
    for(int j=0; j<nElems; j++)         //for each element,
        cout << v[j] << " ";            //display it
    cout << endl;
    }
//--------------------------------------------------------------
};  //end class HighArray
////////////////////////////////////////////////////////////////
int main()
{
    int maxSize = 100;                  //array size
    HighArray arr(maxSize);             //vector
    arr.insert(77);                     //insert 10 items
    arr.insert(99);
    arr.insert(44);
    arr.insert(55);
    arr.insert(22);
    arr.insert(88);
    arr.insert(11);
    arr.insert(0);
    arr.insert(66);
    arr.insert(33);

    arr.display();                      //display items

    int searchKey = 35;                 //search for item
    if( arr.find(searchKey) )
        cout << "Found " << searchKey << endl;
    else
        cout << "Can't find " << searchKey << endl;

    cout << "Deleting 0, 55, and 99" << endl;
    arr.remove(0);                      //delete 3 items
    arr.remove(55);
    arr.remove(99);

    arr.display();                      //display items again
    return 0;
}  //end main()
```

W programie lowArray.cpp kod odpowiedzialny za wyszukanie elementu zajął osiem linijek. W programie highArray.cpp zajął już tylko jedną. Dodatkową korzyścią jest to, że użytkownik klasy (tzn. main()) nie musi pilnować indeksowania tablicy. Co więcej użytkownik klasy nie musi nic wiedzieć o strukturze przechowującej dane użytej w highArray. Struktura ta jest ukryta za interfejsem. W następnym przykładzie, zobaczymy ten sam interfejs z inną strukturą danych.

Abstrakcja

Proces oddzielania *jak* od *co* (jak operacje są wykonywane w klasie w przeciwieństwie co jest widziane przez użytkownika klasy) nazywa się abstrakcją.

Przykład 5 (C++) Przykład klasy z tablicą zawierająca uporządkowane dane. W przykładzie tym do odnajdowania elementu zastosowane zostało przeszukiwanie binarne (tylko w przypadku posortowanych danych).

[orderedArray.cpp]

```
//orderedArray.cpp
//demonstrates ordered array class
#include <iostream>
#include <vector>
using namespace std;
////////////////////////////////////////////////////////////
class OrdArray
{
    private:
        vector<double> v;                 //vector v
        int nElems;                       //number of data items
    public:
//-----------------------------------------------------------
    OrdArray(int max) : nElems(0)         //constructor
    { v.resize(max); }                    //size the vector
//-----------------------------------------------------------
    int getSize()                         //return number of
    { return nElems; }                    //elements
//-----------------------------------------------------------
    int find(double searchKey)
    {
        int lowerBound = 0;
        int upperBound = nElems-1;
        int curIn;

        while(true)
        {
            curIn = (lowerBound + upperBound ) / 2;
            if(v[curIn]==searchKey)
                return curIn;                 //found it
            else if(lowerBound > upperBound)
                return nElems;                //can't find it
            else                              //divide range
            {
                if(v[curIn] < searchKey)
                    lowerBound = curIn + 1; //it's in upper half
                else
                    upperBound = curIn - 1; //it's in lower half
            }  //end else divide range
        }  //end while
    }  //end find()
//-----------------------------------------------------------
    void insert(double value)             //put element into array
    {
        int j;
        for(j=0; j<nElems; j++)           //find where it goes
            if(v[j] > value)              //(linear search)
                break;
        for(int k=nElems; k>j; k--)       //move bigger ones up
            v[k] = v[k-1];
        v[j] = value;                     //insert it
        nElems++;                         //increment size
    }  //end insert()
//-----------------------------------------------------------
```

```
    bool remove(double value)
    {
        int j = find(value);
        if(j==nElems)                           //can't find it
            return false;
        else                                    //found it
        {
            for(int k=j; k<nElems; k++)         //move bigger ones down
                v[k] = v[k+1];
            nElems--;                           //decrement size
            return true;
        }
    }  //end remove()
//--------------------------------------------------------------
    void display()                              //displays array contents
    {
        for(int j=0; j<nElems; j++)             //for each element,
            cout << v[j] << " ";                //display it
        cout << endl;
    }
//--------------------------------------------------------------
};  //end class OrdArray
////////////////////////////////////////////////////////////////
int main()
{
    int maxSize = 100;              //array size
    OrdArray arr(maxSize);          //create the array

    arr.insert(77);                 //insert 10 items
    arr.insert(99);
    arr.insert(44);
    arr.insert(55);
    arr.insert(22);
    arr.insert(88);
    arr.insert(11);
    arr.insert(00);
    arr.insert(66);
    arr.insert(33);

    int searchKey = 55;             //search for item

    if( arr.find(searchKey) != arr.getSize() )
        cout << "Found " << searchKey << endl;
    else
        cout << "Can't find " << searchKey << endl;

    arr.display();                  //display items

    cout << "Deleting 0, 55, and 99" << endl;
    arr.remove(00);                 //delete 3 items
    arr.remove(55);
    arr.remove(99);

    arr.display();                  //display items again
    return 0;
}  //end main()
```

Korzyść z stosowania uporządkowanych tablic

Główną korzyścią jest szybsze wyszukiwanie. Jednakże tracimy na szybkości wstawiania nowych danych do tablicy (dane o większych wartościach muszą być przesunięte w górę

tablicy w celu zrobienia miejsca na nową wartość).

Przykład 6 (C++) Przechowywanie obiektów. Implementacja klasy Person. Przykład klasy mogącej posłużyć do przechowywania danych personelu.

[orderedArray.cpp]

```
//classDataArray.cpp
//data items as class objects
#include <iostream>
#include <string>
#include <vector>
using namespace std;
////////////////////////////////////////////////////////////////
class Person
{
    private:
        string lastName;
        string firstName;
        int age;
    public:
//-----------------------------------------------------------
    Person(string last, string first, int a) :   //constructor
    lastName(last), firstName(first), age(a)
    {  }
//-----------------------------------------------------------
    void displayPerson()
    {
        cout << "   Last name: " << lastName;
        cout << ", First name: " << firstName;
        cout << ", Age: " << age << endl;
    }
//-----------------------------------------------------------
    string getLast()                    //get last name
    { return lastName; }
};  //end class Person
////////////////////////////////////////////////////////////////
class ClassDataArray
{
    private:
        vector<Person*> v;               //vector of pointers
        int nElems;                      //number of data items
    public:
//-----------------------------------------------------------
    ClassDataArray(int max) : nElems(0) //constructor
    { v.resize(max); }                  //create the array
//-----------------------------------------------------------
    ~ClassDataArray()                   //destructor
    {
        for(int j=0; j<nElems; j++)     //delete each element
            delete v[j];
    }
//-----------------------------------------------------------
    Person* find(string searchName)
    {                                   //find specified value
        int j;
        for(j=0; j<nElems; j++)         //for each element,
            if( v[j]->getLast() == searchName )  //found item?
                break;                  //exit loop before end
```

```
        if(j == nElems)                 //gone to end?
            return NULL;                //yes, can't find it
        else
            return v[j];                //no, found it
    };  //end find()
//--------------------------------------------------------------
    //put person into array
    void insert(string last, string first, int age)
    {
        v[nElems] = new Person(last, first, age);
        nElems++;                       //increment size
    }
//--------------------------------------------------------------
    bool remove(string searchName)      //delete person from array
    {
        int j;
        for(j=0; j<nElems; j++)         //look for it
            if( v[j]->getLast() == searchName )
                break;
        if(j==nElems)                   //can't find it
            return false;
        else                            //found it
        {
            delete v[j];                //delete Person object
            for(int k=j; k<nElems; k++) //shift down
                v[k] = v[k+1];
            nElems--;                   //decrement size
            return true;
        }
    }  //end remove()
//--------------------------------------------------------------
    void displayA()                     //displays array contents
    {
        for(int j=0; j<nElems; j++)     //for each element,
            v[j]->displayPerson();      //display it
    }
//--------------------------------------------------------------
};  //end class ClassDataArray
////////////////////////////////////////////////////////////////
int main()
{
    int maxSize = 100;                  //array size
    ClassDataArray arr(maxSize);        //array

    arr.insert("Evans", "Patty", 24);   //insert 10 items
    arr.insert("Smith", "Lorraine", 37);
    arr.insert("Yee", "Tom", 43);
    arr.insert("Adams", "Henry", 63);
    arr.insert("Hashimoto", "Sato", 21);
    arr.insert("Stimson", "Henry", 29);
    arr.insert("Velasquez", "Jose", 72);
    arr.insert("Lamarque", "Henry", 54);
    arr.insert("Vang", "Minh", 22);
    arr.insert("Creswell", "Lucinda", 18);

    arr.displayA();                     //display items

    string searchKey = "Stimson";       //search for item
    cout << "Searching for Stimson" << endl;
    Person* found;
    found=arr.find(searchKey);
```

```
    if(found != NULL)
    {
        cout << "   Found ";
        found->displayPerson();
    }
    else
        cout << "   Can't find " << searchKey << endl;

    cout << "Deleting Smith, Yee, and Creswell" << endl;
    arr.remove("Smith");                //delete 3 items
    arr.remove("Yee");
    arr.remove("Creswell");

    arr.displayA();                     //display items again
    return 0;
}  //end main()
```

Opracowano na podstawie:
Robert Lafore – "Teach Yourself Data Structures And Algorithms In 24 Hours"