Programowanie komputerowe

Zajęcia 6

Tablice wielowymiarowe

- Tablicę dwuwymiarową możemy deklarować statycznie:

  ```
  typ nazwa[m][n];
  ```

  Ma ona elementy `nazwa[i][j]` dla `i=0,...,m-1`, `j=0,...,n-1`.
- Dynamicznie: dwuwymiarowa tablica jest skonstruowana jako tablica wskaźników do tablic jednowymiarowych, np. tak:

```
int** t;
t=new int*[m];
for(int i=0; i<m; i++)
    t[i]=new int[n];
```

Tablice wielowymiarowe (2)

- Żeby usunąć tablicę, musimy usunąć najpierw tablice z wartościami, a następnie tablicę wskaźników:

```
for(int i=0; i<m; i++)
    delete[] t[i];
delete t;
```

- Analogicznie tworzy się tablice o większej ilości wymiarów.
- Można tworzyć dynamicznie tablice, które nie są kwadratowe – do tablicy ze wskaźnikami można podpiąć tablice różnej długości.

Klasa `string`

Przechowywanie napisów jako ciągów znaków zakończonych zerem nie zawsze jest wygodne. Alternatywną metodą jest użycie klasy `string`.

Tworzenie napisów:

```
string s;  // tworzy pusty napis ""
string s("Napis"); // tworzy napis "Napis"
string s(7, 'A');  // tworzy napis "AAAAAAA"
string s(t, 5, 3); // t powinno być napisem, s stanie się równe
                   //  podciągowi t od piątego znaku o długości max 3
```

Klasa `string` (2)

- Drukowanie – przy pomocy `cout`.
- Wczytywanie: można użyć `cin >> s`, ale lepiej

  ```
  getline(cin, s);
  ```

  wczytuje całą linię z `cin` i zapisuje w `s`.
- Dostęp do poszczególnych znaków: przy pomocy []

  ```
  s[2]='A';
  ```

  lub przy pomocy metody at:

  ```
  s.at(2)='A';
  ```

  Ta druga metoda jest bezpieczniejsza – program sprawdzi czy indeks jest poprawny.

Klasa `string` – metody

Zmienne typu string to obiekty, tj. takie zmienne mogą wykonywać funkcje (nazywane też metodami). Metody dla obiektu s wywołujemy

```
s.nazwa_funkcji(parametry)
```

Przykładem metody klasy `string` jest `at` z poprzedniej strony. Inne są wymienione poniżej:

- `size_t length()`
  Zwraca długość napisu. `size_t` oznacza typ służący do przechowywania liczb naturalnych (można zakładać, że jest to `int`). Przykład użycia:
  `cout << s.length();`

Klasa `string` – metody (2)

- `size_t size()`
  Zwraca długość, synonim `length`.
- `string& append(const string& str)`
  dołącza parametr na końcu napisu. Zwraca napis dla którego wywołaliśmy metodę – dzięki temu można napisać np.

  ```
  s.append("Abc").append("def");
  ```

  Modyfikator `const` w przy parametrze oznacza, że funkcja na pewno nie zmieni parametru (przekazanego przez zmienną). Można też użyć operatora +=, np. tak:

  ```
  s+="Abcdef";
  ```

Klasa `string` – metody (3)

- `string& insert(size_t pos, const string& str)`
  Wstawia napis `str` przed znakiem o indeksie `pos`.
- `string substr(size_t pos=0, size_t len=npos)`
  Tworzy nowy napis który zawiera podciąg od pozycji `pos` o długości (co najwyżej) `len`. Jeśli pominiemy drugi parametr, kopiuje się podciąg do końca napisu. Jeśli pominiemy oba, tworzy się kopia. Przykład:

```
string s("ABCDEF");
cout << s.substr(2,2) << endl << s.substr(3) << endl;
```

  drukuje

```
CD
DEF
```

Metody – jak je znaleźć

Klasa `string` posiada jeszcze wiele innych metod. Nie warto się ich uczyć na pamięć, ale trzeba wiedzieć jak je znaleźć.

- w książce
- w dokumentacji (w Code::Blocks nie ma, ale jest np. w Visual Studio)
- w internecie, np. na www.cplusplus.com

Klasa `vector`

Klasa `vector` umożliwia obsługę tablic. Tak naprawdę `vector` nie jest klasą, ale wzorcem umożliwiającym tworzenie klas uzależnionych od typu elementów, tak więc mamy np.

- `vector<int>` – tablica liczb typu int
- `vector<string>` – tablica napisów
- `vector<vector<int> >` – tablica tablic liczb (tablica dwuwymiarowa)

Żeby używać klasy `vector` należy użyć dyrektywy

```
#include <vector>
```

Konstruktory klasy `vector<T>`

- `vector<T>()` – pusta tablica
  przykład: `vector<int> v;`
- `vector<T>(size_t n)` – tablica o zadanej długości
  przykład: `vector<int> v(10);`
- `vector<T>(size_t n, T val)` – tablica o zadanej długości wypełniona wartością
  przykłady:

```
vector<string> v(10, "ABC");
vector<vector <int> > v(5,vector<int>(5,7));
```

- `vector<T>(const vector<T>& v)` – kopia tablicy

Metody klasy `vector<T>`

- `T& at(size_t n)` – element o indeksie n
  przykład: `v.at(5);`
- `T& operator[](size_t n)` – jak wyżej
  przykład `v[5];`
- `vector<T>& operator=(const vector<T>& v)` – przypisanie
  przykład:

```
vector<int> v(5,3);
vector<int> w;
w=v;
```

Metody klasy `vector<T>` (2)

- `size_t size()` – długość tablicy
- `void resize(size_t n)` – zmienia długość tablicy
- `void resize(size_t n, const T& val)` – zmienia długość, dodatkowe elementy ustawia na `val`
- `void push_back(const T& val)` – dodaje element na końcu, tablica się wydłuża o 1
- `void pop_back()` – usuwa ostatni element
- `void clear()` – usuwa wszystkie elementy

Ćwiczenia

1. Napisać funkcję

   ```
   void rejestr()
   ```

   która wczytuje liczby z klawiatury aż zostanie wpisane 0. Następnie drukuje wszystkie te liczby na ekranie.
2. Napisać funkcję

   ```
   vector<int> czynniki(int n)
   ```

   która zwraca tablicę z czynnikami pierwszymi liczby n.
3. Napisać funkcję

   ```
   string odwroc(string s)
   ```

   która zwraca napis czytany od końca.

Ćwiczenia (2)

4. Napisać program, który prosi użytkownika o wpisanie tekstu. Tekst może składać się z wielu linii; wczytywanie kończy się gdy zostanie wpisana pusta linia. Następnie należy:
   a. wydrukować wpisany tekst
   b. wypisać liczbę wpisanych linii
   c. wypisać liczbę wpisanych wyrazów; wyraz to ciąg znaków pomiędzy spacjami
   d. wydrukować listę znaków, które występują w tekście przynajmniej raz