Programowanie komputerowe

Zajęcia 5

Tablice w stylu C a wskaźniki

- Tablicę można utożsamiać ze wskaźnikiem wskazującym na jej pierwszy element.
- Przypomnienie: dla wskaźników można używać operatorów `++`, `−`, `+`, `−`.
- `++` (`−−`) powoduje przesunięcie wskaźnika o jedno miejsce w prawo (lewo).
- Dodanie (odjęcie) liczby całkowitej powoduje przesunięcie wskaźnika o podaną liczbę miejsc w prawo (lewo).
- Można odejmować wskaźniki jeśli wskazują na tą samą tablicę.

Wskaźniki i tablice – przykład

```
int main() {
    int t[5]={2,4,7,5,7};
    for(int* p=t; p<t+5; p++)
        cout << *p << endl;
}
```

Ten program drukuje zawartość tablicy na ekranie.

Napisy w stylu C

Konwencja: zapisany w tablicy ciąg znaków zakończony jest znakiem o kodzie 0, można go zapisać jako `0`, `NULL` lub `'\0'` (nie mylić ze znakiem `'0'`).

Przykład: tablica o elementach
`'C'  'z'  'w'  'a'  'r'  't'  'e'  'k'  '\0'  'A'  '?'`
reprezentuje napis `"Czwartek"`. Znaki występujące po znaku `'\0'` nie są istotne.

Dzięki temu, że `0` kończy napis nie ma potrzeby przekazywania jego długości.

Konwencja ta dotyczy **tylko** tablic znaków, które reprezentują napis.

Przykłady funkcji

Funkcja zwracająca długość napisu:

```
int dlugosc(char* s) {
    int i;
    for(i=0; s[i]; i++);
    return i;
}
```

- używamy konwencji, że wyrażenie typu całkowitego lub znakowego jest prawdziwe jeśli jest różne od zera (fałszu)
- jeśli ciało pętli jest puste, możemy użyć średnika zamiast `{}`

Jeszcze raz długość napisu

```
int dlugosc(char* s) {
    char* p=s;
    while(*p++);
    return p-s-1;
}
```

Niezbyt czytelnie ale krótko. W jaki sposób działa ta funkcja?

Napisy – przykład

Funkcja sprawdzająca czy dwa napisy są jednakowe:

```
bool rowne(char* s, char* t) {
    int i=0;
    while(s[i]==t[i]) {
        if(s[i]==0)
            return true;
        i++;
    }
    return false;
}
```

Tablice tworzone dynamicznie

- Podczas poprzednich zajęć zajmowaliśmy się tablicami, których rozmiar był znany w momencie kompilacji (np. wiadomo było, że tablica będzie zawierać 14 elementów).
- Do rozwiązywania problemów programistycznych konieczna jest możliwość tworzenia i niszczenia obiektów w czasie pracy programów.
- Jeśli potrzebujemy tablicy, której dokładny rozmiar nie jest znany w momencie kompilacji (np. zależy od wartości jakiejś innej zmiennej), możemy stworzyć ją dynamicznie.

Tablice tworzone dynamicznie – schemat

1. Tablica będzie potrzebować “gdzieś” znaleźć się w pamięci (mieć adres):
   - Deklarujemy wskaźnik, za pomocą którego będziemy mieć dostęp do tablicy (to będzie też nazwa naszej tablicy).
2. Rozmiar tablicy musi mieć do tego czasu ustaloną wartość.
3. Tworzymy tablicę z wykorzystaniem operatora `new`:
   - Ustawiamy wskaźnik na pierwszy element tablicy – `new` upewni się, że przydzielanie pamięci zakończyło się powodzeniem zanim przekaże adres.

Tablice tworzone dynamicznie – kod

```
//deklaracja wskaznika -- taka nazwe przybierze nasza tablica

int* p;

// zmienna odpowiadajaca za rozmiar musi miec ustawiona wartosc przed jej uzyciem!

int rozmiar = ...;

//tworzenie tablicy -- operator new i ustawienie wskaznika na pierwszy element

p = new int[rozmiar];

// dalej mozemy korzystac z tablicy standardowo: p[0], ... p[rozmiar-1]
```

Niszczenie tablic (czyszczenie pamięci)

- Pamięć alokowana dynamicznie musi zostać na koniec działania programu zwolniona.
- Dotyczy to każdej sytuacji, w której będziemy samodzielnie alokować pamięć o nieznanej wielkości w momencie kompilacji!
- Niszcząc tablicę, której wielkości nie znamy, potrzebujemy rozwiązać dwa problemy:
  - Zniszczyć tablicę
  - Zwolnić wcześniej przydzieloną pamięć

W C++ rozwiązaniem jest operator komplementarny do `new`: `delete`.

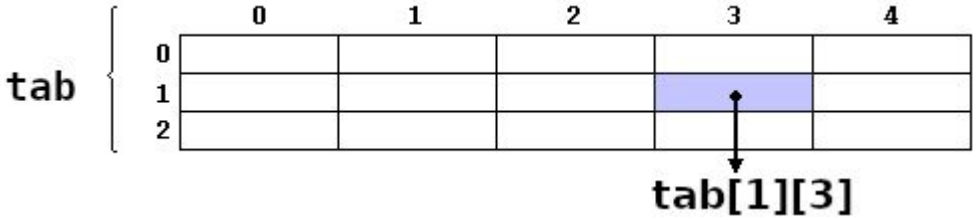
Niszczenie tablic – schemat i kod

- Operator `new` zwracał wskaźnik, dlatego operator `delete` potrzebuje dostać wskaźnik
- Musimy poinformować kompilator, że podany adres jest faktycznie adresem początku tablicy (doklejamy nawiasy kwadratowe do `delete`)
- Nie musimy tutaj podawać rozmiaru – kompilator “znajdzie go”, bo zapisany został podczas tworzenia
- Wywołujemy gdy tablica przestaje nam być potrzebna – tam, gdzie chcemy posprzątać pamięć.
- **`delete[]` używamy wyłącznie dla tablic stworzonych wcześniej z wykorzystaniem `new`**

```
// p to nazwa naszej tablicy
    delete[] p;
```

Tablice wielowymiarowe

- Tablice nie muszą mieć tylko jednego wymiaru. Tablice wielowymiarowe to *tablice tablic.*
- Skupimy się na działaniach na tablicach dwuwymiarowych
- Tablica jednowymiarowa to ustalona liczba kolumn. Dla dwuwymiarowej potrzebujemy “dołożyć” drugi wymiar (liczbę wierszy)
- Konwencja: pierwsza współrzędna to wiersze, druga kolumny



Tablice wielowymiarowe (2)

- Tablicę dwuwymiarową możemy tworzyć statycznie, np. tak:

```
int tab[3][5];
    int tab[3][2] = {{1,3}, {4,5}, {{0,-1}};
```

- Dynamicznie: dwuwymiarowa tablica jest skonstruowana jako tablica wskaźników do tablic jednowymiarowych, np. tak:

```
int** t;
// pamiec na wiersze
t=new int*[w];
// pamiec na kolumny w ramach wiersza
for(int i=0; i<w; i++)
    t[i]=new int[k];
```

Tablice wielowymiarowe (3)

- Jeśli tworzyliśmy tablicę dzięki `new`, musimy ją na koniec usunąć!
- Usuwamy najpierw tablice z wartościami, a następnie tablicę wskaźników (psujemy od wewnątrz do zewnątrz):

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

- Orientacyjna zasada: ile razy użyliśmy `new`, tyle razy potrzebujemy `delete`
- Analogicznie działamy z tablicami o większej liczbie wymiarów.
- Można tworzyć dynamicznie tablice, które nie są kwadratowe – do tablicy ze wskaźnikami można podpiąć tablice różnej długości.

Zadania

Napisać funkcje:

1. `int wystapienia(char* s, char c)`
   która zwraca liczbę wystąpień znaku `c` w napisie `s`.
2. `void drukujWspak(char* s)`
   która drukuje napis od ostatniego znaku do pierwszego.
3. `void drukujBezSpacji(char* s)`
   która drukuje na ekranie napis `s` pozbawiony spacji.
4. `int ileWyrazow(char* s)`
   która zwraca liczbę wyrazów w napisie `s`.

Zadania (2)

5. `int znajdz(char* s, char* t)`
   która zwraca indeks pierwszego wystąpienia napisu `t` w `s` (lub `-1` jeśli nie ma wystąpień).
6. `int liczba(char* s)`
   która zwraca wartość liczbową napisu `s` (np. `liczba("376")`=376).
7. `bool anagramy(char* s, char* t)`
   która zwraca true jeśli `s` i `t` to anagramy.

Zadania (3*)

8. `void wypelnij_los(int** t, int w, int k)`
która wypełni tablicę dwuwymiarową liczbami pseudolosowymi (argumentami funkcji są tablica dwuwymiarowa, liczba wierszy, liczba kolumn)
Wskazówka: Skorzystaj z funkcji rand() z biblioteki <cstdlib> (sprawdź dokumentację, np. tutaj).

9. `void transpozycja(int** t, int n)`
która wykona transpozycję zadanej macierzy kwadratowej (można dla testów wypełnić ją, wykorzystując Z8).

10**. Zaimplementować Spiralę Ulama dla liczb od 1 do 100.