Politechnika Wrocławska

Struktury danych, unie

wer. 8 z drobnymi modyfikacjami!

Wojciech Myszka

Katedra Mechaniki, Inżynierii Materiałowej i Biomedycznej

2023-04-24 10:25:18 +0200

Struktury danych

1. W Pascalu — rekordy.
2. Struktura „podobna” do tablicy, ale pozwalająca na przechowywanie danych różnych typów.
3. Przykłady:
   - Lista płac:
     - Imię
     - Nazwisko
     - PESEL
     - Numer konta bankowego
     - Kwota do wypłaty
     - ...
   - Współrzędne punktu:
     - współrzędna X
     - współrzędna Y
     - (ewentualnie) współrzędna Z

Zacznijmy od przykładu

Ułamki

Politechnika Wrocławska

Zadanie

- Chcemy (musimy?) napisać program–kalkulator wykonujący obliczenia na ułamkach „zwykłych”.
- Ułamki takie to liczby postaci:

$$\frac{p}{q}$$

  gdzie $p, q \in N$ oraz $q \neq 0$.
- Kalkulator będzie wykonywał operacje dodawania, odejmowania, mnożenia i dzielenia.
- Do każdej operacji musimy napisać funkcję.
- Dodatkowo potrzebne będą funkcje upraszczania ułamków i ich drukowania. Chcemy żeby wydruk ułamków wyglądał jak najbardziej naturalnie:

```
     17
-35---
    123
```

Sposób przechowywania danych

1. Pierwszą sprawą wymagającą naszej uwagi jest sposób przechowywania danych.
2. Najwygodniej (czemu?) przechowywać wszystkie dane, na których prowadzimy operacje jako ułamki (niewłaściwe)
3. Wydaje się, że najwygodniej będzie dane przechowywać jako dwie liczby:
   - licznik przechowujący również znak liczby
   - mianownik
4. Po każdej operacji ułamek będzie normalizowany (upraszczany).

Politechnika Wrocławska

Operacje

- dodawanie/odejmowanie

$$\frac{a}{b} \pm \frac{c}{d} = \frac{a * d \pm c * b}{b * d}$$

- mnożenie

$$\frac{a}{b} * \frac{c}{d} = \frac{a * c}{b * d}$$

- dzielenie

$$\frac{a}{b} / \frac{c}{d} = \frac{a * d}{b * c}$$

Uwagi

Ponieważ licznik przechowuje znak — w wyniku dzielenia może okazać się, że mianownik stanie się ujemny. Trzeba to sprawdzić i skorygować.
Do rozstrzygnięcia pozostaje kwestia działań podejmowanych gdy mianownik będzie równy zeru.

Politechnika Wrocławska

Naiwna próba realizacji

- Napiszmy funkcję suma.
- argumenty: wartości a, b, c, d (zmienne typu **int**)
- wynik: nie będzie mógł być przekazany poleceniem **return** — można tak przesłać tylko jedną zmienną
  czyli:
  - albo piszemy dwie funkcje do wyliczania wartości licznika i mianownika będziemy potrzebowali cztery funkcje do wyliczania liczników dla sumy, różnicy, iloczynu i ilorazu oraz dwie funkcje do wyliczania mianownika: jedną dla sumy, różnicy i iloczynu, oraz drugą dla ilorazu;
  - albo wyniki przekazujemy „przez adres” jako wskaźniki

  Drugie rozwiązanie wydaje się bardziej racjonalne

Politechnika Wrocławska

Suma

```
void suma ( int a, int b, int c, int d, int *p, int *q )
{
 *p = a * d + c * b;
 *q = b * d;
}
```

Użycie:

```
int a, b, c, d, e, f;
...
suma (a, b, c, d, &e, &f);
```

Politechnika Wrocławska

Czy można inaczej?

1. Możliwość posiadania zmiennej, która będzie mogła przechowywać więcej niż jedną wartość ułatwiłaby nasz problem.
2. Taki typ zmiennych istnieje w języku C i nazywa się **strukturą**.
3. Deklarujemy najpierw strukturę danych:

```
struct Ulamek
{
 int l;
 int m;
};
```

4. Używamy polecenia **typedef** aby łatwiej używać zmiennych

```
typedef struct Ulamek ulamek;
```

i możemy teraz deklarować zmienne w sposób następujący:

```
ulamek A, B, C;
```

Politechnika Wrocławska

Zalety i wady takiego postępowania

+ Mamy zmienną mogącą przechowywać więcej niż jedną wartość.
+ Przechowywane wartości mogą być różnego typu.
+ Ponieważ korzystamy z nowego typu — funkcje mogą używać tego typu we wszelkich obliczeniach.
– Nie można wykonywać operacji na takich zmiennych złożonych.
– Dostęp do składowych struktury jest nieco bardziej skomplikowany;
– Nie można (łatwo) nadawać wartości zmiennym tego typu (nie istnieją stałe typu strukturalnego).

Politechnika Wrocławska

Nowe sformułowanie funkcji suma

```
struct Ulamek
{
 int l;
 int m;
};
typedef struct Ulamek ulamek;
ulamek suma ( ulamek A, ulamek B )
{
 ulamek Wynik;
 Wynik.l = A.l * B.m + B.l * A.m;
 Wynik.m = A.m * B.m;
 return Wynik;
}
...
ulamek aa, bb, cc;
...
cc = suma ( aa, bb );
```

Politechnika Wrocławska

Co dalej?

1. Możemy wykonywać operacje złożone (**ulamek** aa, bb, cc;):

```
cc = iloczyn(suma(aa, bb), roznica(aa, bb))
```

2. Deklarując zmienną, można nadać jej wartość (ale tylko tu!):

```
ulamek aa = {1, 2}; bb = {3, 4}, cc;
```

3. Ma sens coś takiego:

```
int z = suma(aa, bb).l;
```

Politechnika Wrocławska

Struktury danych

Deklaracja

1. Deklaracja struktury, tak na prawdę, to deklaracja nowego typu danych!

```
struct point {
    int x;
    int y;
};
```

2. x i y to **składowe** struktury.
3. Deklaracja zmiennej:

```
struct point punkt1, punkt2, punkt3, pt;
```

4. Można też tak:

```
struct point3D {
    int x;
    int y;
    int z;
} p1, p2, p3;
```

   Można też w dwu etapach (jak w ułamku):

   4.1 najpierw definiujemy strukturę,
   4.2 później definiujemy nowy typ używając **typedef**.

Struktury

Dane

1. Inicjalizacja:

```
struct point maxpt = { 320, 200 };
```

2. W wyrażeniach dostęp uzyskuje się *nazwa_struktury.składowa*
3. Kropka to **operator** składowej struktury.
4. Przykłady:

   4.1
   ```
   printf("%d,%d", punkt1.x, punkt1.y);
   ```

   4.2 Odległość między punktami

   ```
   double dist, sqrt(double); /* sqrt: pierwiastek */
   dist = sqrt((double)pt.x * pt.x + (double)pt.y * pt.y);
   ```

Struktury

Przykład cd

- Zagnieżdzona struktura

```
struct rect {
    struct point pt1;
    struct point pt2;
};
```

Gdy zadeklarujemy screen:

```
struct rect screen;
screen.pt1.x = 100;
```

nadaje wartośc współrzędnej x pierwszego punktu (pt1) struktury ekran

- Inicjacja struktury złożonej

```
struc rect ekran = { { 0, 0 }, { 1024, 768 }};
```

Struktury i funkcje I

1. Funkcja może zwracać daną typu strukturalnego.

```
/* makepoint: utworz punkt ze
              wspolrzednych x i y */
struct point makepoint(int x, int y)
{
    struct point temp;
    temp.x = x;
    temp.y = y;
    return temp;
}
```

   Użycie punkt2 = makepoint(100, 59);

2. Parametrami funkcji mogą być dane typu strukturalnego

Struktury i funkcje II

```
/* addpoint: dodaj dwa punkty */
struct point addpoint(struct point p1, \
                      struct point p2)
{
    p1.x += p2.x;
    p1.y += p2.y;
    return p1;
}
```

użycie:

```
int main (void)
{
        struct rect ekran = { { 0, 0 }, { 1024, 768 } };
        struct point srodek;
        srodek = makepoint(addpoint(ekran.pt1,
                           ekran.pt2).x/2,\
                           addpoint(ekran.pt1,
                           ekran.pt2).y/2);
        printf("(%d,%d)", srodek.x, srodek.y);
```

Struktury i funkcje III

```
        return 0;
}
```

Politechnika Wrocławska

Struktury i wskaźniki I

1. W sytuacji gdy do funkcji jako daną mamy przekazać bardzo rozbudowaną strukturę (będzie ona **kopiowana** do zmiennej tymczasowej) lepiej jest użyć wskaźnika.
2. Deklaracja jak zwykle:

```
struct point origin, *pp;
```

   origin to struktura (o składowych x i y), pp to wskaźnik do struktury typu point; (*pp).x oraz (*pp).y to jej składowe.

3. Użycie

```
pp = &origin;
printf("punkt poczatkowy (%d,%d)\n", \
                    (*pp).x, (*pp).y);
```

Politechnika Wrocławska

Struktury i wskaźniki II

4. Ponieważ struktur używa się bardzo często podobnie jak wskaźników do nich, zeby uprościć życie wymyślono specjalną notację. Jeżli `p` jest wskaźnikiem do struktury to aby się dobrać do składowej, używamy takiej notacji: `p->skladowa_struktury`. Zatem poniższy zapis jest równoważny poprzedniemu:

```
printf("punkt poczatkowy (%d,%d)\n", \
                          pp->x, pp->y);
```

5. W przypadku struktur zagnieżdżonych sprawa nieco się komplikuje (to samo można zapisać na kilka różnych sposobów!):

```
struct rect r, *rp = &r;
```

Następujące wyrażenia są równoważne:

```
r.pt1.x
rp->pt1.x
(r.pt1).x
(rp->pt1).x
```

Struktury i wskaźniki III

6. Operatory strukturowe ., –> oraz nawiasy okrągłe () wywołania funkcji oraz nawiasy kwadratowe [] indeksowania tablicy mają najwyższy priorytet (najsilniej wiążą swoje argmenty).

Politechnika Wrocławska

Tablice struktur

1. Podobnie jak z danych innych typów, również ze struktur można tworzyć tablice (bazy danych???)

2. ```
   struct point dane[100];
   ```

   deklaruje tablicę o 100 elementach, z których każdy to dana typu point — struktura złożona z dwu składowych x i y.
   ```

Politechnika Wrocławska

Tablice struktur

Przykład

1. Piszemy program zliczający częstość występowania słów kluczowych języka C.
2. Dane: tablica, każdym jej elementem jest para (słowo kluczowe, liczba wystąpień) — struktura.
3. Schemat blokowy:
   - 3.1 Jeżeli koniec pliku — **skończ**.
   - 3.2 Wprowadź słowo (z pliku).
   - 3.3 Znajdź słowo w danych.
   - 3.4 Jezeli wystąpiło — zwiększ odpowiedni licznik.
   - 3.5 (Jeżeli nie wystąpiło — przechodzimy dalej.)
   - 3.6 Przejdź na początek.

Sam program znajduje się w [1]:


B. W. Kernighan, D. M. Ritchie.
Język ANSI C.
WNT, Warszawa, 2007.

Tablice struktur

Przykład — dane

1. Zakładamy, że słowa kluczowe uszeregowane są w kolejności rosnącej — ułatwi to wyszukiwanie.
2. Do wyszukiwania wykorzystamy metody szukania binarnego (podział na pół).
3. Ile miejsca w pamięci (bajtów) zajmuje struktura key?

```
struct key {
    char *word;
    int count;
} keytab[] = {
    "auto", 0,
    "break", 0,
    "case", 0,
    "char", 0,
    "const", 0,
    /* ... */
    "volatile", 0,
    "while", 0
};
```

Pola bitowe

1. Pamięć jest tania i zazwyczaj nie ma na potrzeby jej oszczędzać.
2. Czasami chcemy jednak operować an bitach.
3. Definiujemy sobie coś takiego:

```
struct {
    unsigned int is_keyword : 1;
    unsigned int is_extern  : 1;
    unsigned int is_static  : 1;
} flags;
```

   ta 1 (po dwukropku) do długość pola w bitach.
4. Dostajemy zmienną o nazwie flags zawierającą trzy jednobitowe pola.
5. Bardzo łatwo możemy manipulować bitami (zerować je, ustawiać i sprawdzać ich wartość), na przykład:

```
flags.is_extern = flag.is_static = 0;
```

Politechnika Wrocławska

Unie

1. Na pierwszy rzut oka deklaracja unii jest bardzo podobna do deklaracji struktury:

```
union uni {
    int val;
    float fval;
    char sval[4];
} zmienna;
```

2. Podobieństwo jest jednak złudne.
3. Unia to obszar pamięci, w którym zapisać można wartości różnych typów; kompilator dba o przydział odpowiednio dużego obszaru pamięci (mieszczącego „największy” z podnych typów).
4. Użycie: `zmienna.val = 5` albo `x = zmienna.fval`
5. Wykorzystanie...

Politechnika Wrocławska

Deklaracja nowego typu I

1. Standardowy zestaw typów w języku C jest niezbyt bogaty (**int**, **float**, **double**, **char**, **struct**).
2. Niektóre typy mogą być modyfikowane za pomocą słów **short**, **long**, **unsigned**.
3. Język C pozwala na definiowanie „aliasów" dla struktur istniejących (ułatwiają one konstruowanie programów). Służy do tego instrkcja **typedef**:

   ```
   typedef int Lenght;
   ```

   deklarująca **nowy** typ danych o nazwie **Lenght** (służący do deklarowania wszystkich zmiennych związanych z długością):

   ```
   Lenght len, maxlen, minlen;
   ```

4. Możemy w ten sposób stworzyć sobie typ zespolony:

   ```
   typedef struct { double r, theta; } Complex;
   ```

Deklaracja nowego typu II

albo

```
typedef struct { double real, imaginary; }
                Complex;
```

5. Albo cośą takiego:

```
typedef char* string;
```

(Do czego może się to przydać? Ale korzystać ostrożnie!)

Politechnika Wrocławska

Deklaracje typów

```
typedef struct {
     char forename[20];
     char surname[20];
     float age;
     int childcount;
     } person;
```

1. Definiujemy nowy typ o nazwie **person**.
2. Jest to struktura danych (rekord?) o zawartości: imię, nazwisko, wiek, liczba potomstwa.
3. Deklaracja poszczegolnych zmiennych staje się bardzo prosta:

```
person Osoba1, Osoba11, Osoba111;
int a, b, c;
double x, y, z;
```