Структура на програма в C

Част 7 - масиви, оператор за индексиране, sizeof оператор

Иван Георгиев, Христо Иванов, Христо Стефанов

Технологично училище "Електронни системи",
Технически университет, София

5 май 2019 г.

Съдържание

Съдържание

Проблем

- При писане на програми, често се налага да се работи с големи обеми от данни
- За да могат да бъдат обработени в C, те трябва да се вкарат в променливи, с които след това може да се работи
- Не е практично един програмист да декларира голямо количество променливи (напр. 10000) една по една
- Дори и да бъдат декларирани, работата с голям брой променливи е трудоемка, дори и за прости програми, тъй като трябва всяка променлива да бъде посочвана чрез нейното име, за да бъде използвана в израз

Проблем

```
int main() {
    int a1; int a2; int a3; int a4; int a5;
    int a6; int a7; int a8; int a9; int a10;

    cin >> a1; cin >> a2; cin >> a3; cin >> a4; cin >> a5;
    cin >> a6; cin >> a7; cin >> a8; cin >> a9; cin >> a10;

    cout << a10; cout << a9; cout << a8; cout << a7; cout << a6;
    cout << a5; cout << a4; cout << a3; cout << a2; cout << a1;
}
```

Фрагмент 1: Програма за извеждане на 10 числа въведени от потребителя в обратен ред

Съдържание

Структури от данни

- *Структурите от данни* представляват данни, *структурирани* (организирани, разположени) в паметта на компютъра по начин, който позволява ефективното им обработване
- Върху структурите от данни могат да се извършват операции, като най-основните операции са прочитане и променяне на данните съдържащи се в структурата
- Различните структури от данни поддържат различни операции и са различно ефективни в изпълнението им
  - Например намирането на най-малкия елемент в някои структури е еднакво бързо, независимо от броя на данните в структурата, но промяната на елемент е бавна и зависи от броя данни
- Поради тази причина, с какви структури от данни работи е най-добре да се работи, зависи от конкретния проблем, който се опитва да реши програмиста

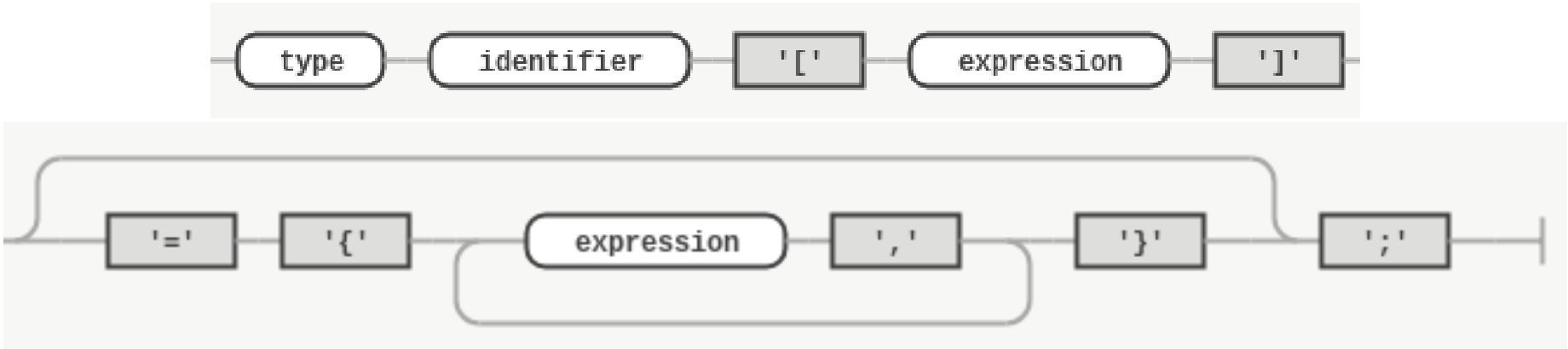
Съдържание

Масиви

- *Масивът* е структура от данни, която разполага данните си последователно в паметта
- Всяка една група от данни (наречена още *елемент на масива*), разполага с уникален пореден номер (*индекс*)
- Масивите поддържат операциите прочитане и промяна на елемент, посочен чрез неговия индекс. Те са еднакво бързи за изпълнение и не зависят от броя елементи в масива
- В C масивите са част от езика и имат специална поддръжка

Масиви

- Както всички данни в C, така и масивите се представят чрез променливи с конкретен тип
- За декларирането на масив е нужно да се посочи типа данни на елементите от масива и техния брой
- Елементите на масива имат произволни стойности, освен ако не бъдат инициализирани чрез *инициализиращ списък*



Фигура 1: Синтактична диаграма на декларация на масив

Декларация на масиви

```c
int a[10];
```

Фрагмент 2: Пример за декларация на масив с 10 елемента

```c
int b[3] = {1, 2 + 4, 3};
```

Фрагмент 3: Пример за декларация на масив с 3 елемента с инициализиращ списък

Инициализиране на масив

- Инициализиращият списък може да съдържа един или повече на брой изрази, но броят на изразите не трябва да превишава броят елементи в масива
- Поредният израз от списъка служи за инициализация на поредния елемент от масива (т.е. първият елемент от масива се инициализира с резултата от първия израз в инициализиращия списък, вторият елемент с втория израз и т.н.)
- В случая, когато в инициализиращият списък има по-малко изрази, отколкото елементи в масива, оставащите неинициализирани елементи имат стойност 0

```
int a[3] = {1, 2, 3}; // OK
int b[3] = {1}; // OK, elements with indices 1 and 2 are '0'
int c[3] = {1, 2, 3, 4}; // ERROR
```

Фрагмент 4: Инициализиране на масив

Инициализиране на масив

- При инициализиране на масив може да се изпусне посочването на общия брой елементи в масива
- В такъв случай масивът ще има толкова елементи, колкото изрази в инициализиращия списък

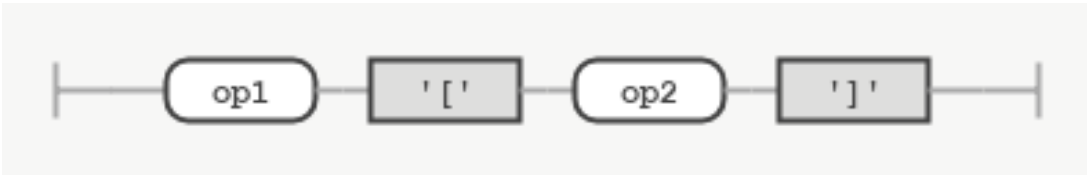
```
/* 'a' and 'b' have 3 elements each, 'c' has 4 elements */
int a[3] = {1, 2, 3};
int b[] = {1, 2, 3};
int c[4] = {1, 2, 3};
```

Фрагмент 5: Инициализация на масив без посочване на броя елементи

Съдържание

Оператор за индексиране

- Елементите от масива могат да бъдат достъпвани чрез *оператора за индексиране*
- Операторът за индексиране създава *временен идентификатор* за елемент, посочен чрез неговия индекс



Фигура 2: Синтактична диаграма на оператор за индексиране

- Първият операнд е име на масив, а вторият - целочислен израз, чиято стойност се използва за индекс. Изразът се пресмята по време на изпълнение на програмата
- В C индексите на елементите на един масив започват винаги от 0 и растат в положителна посока (т.е. първият елемент има индекс 0, вторият - 1, третият - 2, и т.н)

Оператор за индексиране

- Тъй като операторът за индексиране създава временен идентификатор за елемента, неговият резултат може да участва в операции за присвояване (т.е. е `lvalue` стойност)

```c
int main() {
    int a[3] = {10, 20, 30};
    int i = 1;

    printf("%d\n", a[0]); // prints '10'
    printf("%d\n", a[i]); // prints '20'
    printf("%d\n", a[i - 1]); // prints '10'

    for (i = 0; i < 3; i++) {
        a[i] = i; /* OK, 'a[i]' is lvalue and
                     assigning new values is allowed */
    }
}
```

Фрагмент 6: Оператор за индексиране

Съдържание

Оператор sizeof

- В езика C не са посочени големините в байтове на отделните типове, освен на един - типът `char` (1 byte)
- От една страна това позволява езикът да може да се използва от процесори с различни инструкции и архитектури, но от друга страна затормозява писането на програми, тъй като в много случаи има нужда да се знае, колко байта би заела променлива от някакъв тип
- Тъй като големината на отделните типове зависи от това за какъв процесор се компилира програмата, по време на компилация компилаторът знае колко байта памет заема променлива от някой тип
- За да се реши проблема е въведен *оператор sizeof*, чрез който може да се запита компилатора по време на компилиране, колко байта памет би заела променлива от някакъв тип

Оператор sizeof

```
sizeof(op1)
```

Фрагмент 7: Оператор sizeof

- Операндът може да е както само име на тип, така и израз, върху чийто тип да се извърши операцията
- Резултатът от операцията е броя байтове, който би заела променлива от типа, посочен от операнда

```
int main() {
    int c = 1;
    printf("%d\n", sizeof(int)); // prints '4' on most computers
    printf("%d\n", sizeof(c)); // 'c' has type int, prints as above
}
```

Фрагмент 8: Примери за оператор sizeof

Оператор sizeof

```c
int main() {
    int c = 1;
    double a = 2;
    int q[10];

    printf("%d\n", sizeof(c + 1));  /* 'c + 1' has type int
                                       prints 4 on most computers */
    printf("%d\n", sizeof(c / a));  /* 'c / a' has type double
                                       prints 8 on most computers */
    printf("%d\n", sizeof(q));  /* 'q' has type int[10]
                                   prints 40 on most computers */

    printf("%d\n", sizeof(q) / sizeof(q[0]));  /* 'q' has type int[10],
                                                  'q[0]' has type int,
                                                  prints 10 on ALL computers */
}
```

Фрагмент 9: Примери за оператор sizeof