Структура на програма в C

Част 8 - указатели

Иван Георгиев, Христо Иванов, Христо Стефанов

Технологично училище "Електронни системи",
Технически университет, София

13 май 2019 г.

Съдържание

Съдържание

Проблем

- При писане на програми често се налага смяната на стойностите на две променливи
- Тъй като е добре да не се повтаря код е разумно да се направи функция, която да разменя стойностите на две променливи

Проблем

- Проблемът е, че функциите в C винаги използват *стойностите* на *подадените аргументи* за *начални стойности* на своите *параметри* и след това работят върху инициализираните параметри

```c
int swap(int a, int b) {
    int t = a;
    a = b;
    b = t;
    printf("%d\n", a); // prints '5'
    printf("%d\n", b); // prints '10'
    return 0;
}
```

```c
int main() {
    int a = 10;
    int b = 5;
    swap(a, b);
    printf("%d\n", a); /* prints '10',
                          not '5' */
    printf("%d\n", b); /* prints '5',
                          not '10' */
    return 0;
}
```

Фрагмент 1: swap функцията разменя стойностите на параметрите си, а не на подадените като аргументи променливи от main функцията - те остават непроменени

Съдържание

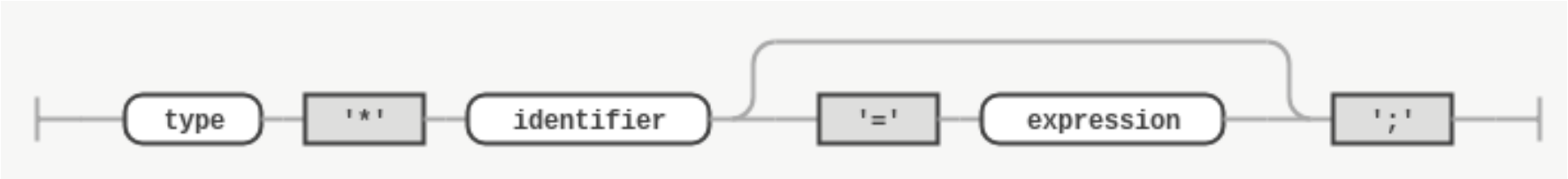
Указатели

- В езика C съществува тип данни, наречен *указател* (*pointer type*)
- Променливите от тип указател, приемат за стойности конкретни места от паметта на компютъра, посочени чрез техните адреси и типа на данните, които съхраняват
- Казва се, че променливите от тип указател "указват" или "сочат" място от паметта на компютъра. За краткост самите променливи от тип указател се наричат *указатели*
- Един указател може да посочва само места, които съхраняват конкретен тип данни, който се определя при декларирането на указателя. Този тип се нарича "сочен" или "указан" тип

```
int *p;  /* pointer to random place in memory
            storing data with 'int' data type */
double *q; /* pointer to random place in memory
            storing data with 'double' data type */
```

Фрагмент 2: Примери за декларации на указатели

Указатели



Фигура 1: Синтактична диаграма на декларация на указател

- Указателите могат да бъдат пренасочвани (т.е. след като са декларирани могат да приемат за стойности и други места чрез оператора за присвояване)
- Работата с указатели е тясно свързана с оператора за взимане на адрес и оператора за индирекция

Съдържание

Оператор за взимане на адрес

- Мястото в паметта на всяка една променлива в C програма, може да бъде взето чрез *оператора за взимане на адрес*
- Върнатият резултат от оператора е мястото в паметта на променливата, посочена от подадения идентификатор - т.е. нейният адрес
- Въпреки че адресът е целочислено число, той не може да се използва като такова, защото има тип на указател, посочващ типа на подадения идентификатор
  - т.е. ако операторът се приложи на променлива от тип `int`, върнатият резултат е от тип `int*`
- Върнатият адрес винаги е насочен към първия (от ляво на дясно) байт от паметта за променливата

Фигура 2: Синтактична диаграма на оператор за взимане на адрес

Оператор за взимане на адрес

```c
int a = 10;
int b[5] = {0};
double d = 1.2;

int *ptr = &a; /* initializes 'ptr' with
                  the location (address) of 'a'
                  using the 'address-of' operator */

ptr = &(b[0]); /* 'address-of' works for temporary
                  identifiers too! */

ptr = &b[0]; /* same as above, [] have higher precedence,
                and () are not needed */

ptr = &d; /* '&d' is an address of 'double' and has type 'double*',
             and should not be assigned to a pointer
             that points to 'int'. "Here be dragons" */
```

Фрагмент 3: Примери с оператор за взимане на адрес

Съдържание

Оператор за индирекция

- *Операторът за индирекция* служи за достъпване на посочената от указател променлива
- Резултатът от операцията е временен идентификатор (`lvalue`) към посочената от указателя променлива
- Това позволява посочената променлива да бъде прочитана и променяна
- При използване на оператора за индирекция, за тип на текущо посочената променлива се счита "сочения" тип от указателя дори и посочената променлива да е от друг тип

Фигура 3: Синтактична диаграма на оператор за индирекция

Оператор за индирекция

```c
int main() {
    int a = 10;
    int *ptr = &a; /* stores the location of 'a' in the
                      'ptr' variable of type pointer
                     using the 'address-of' operator */
    printf("%d\n", *ptr); /* using the indirection operator
                             a temporary identifier is created to 'a',
                             so the value of the expression is
                             the value of 'a',
                             prints '10' */
    return 0;
}
```

Фрагмент 4: Пример за работа с указатели

Оператор за индирекция

```c
int main() {
    int a = 5;
    int b = 6;

    int *ptr = &a; // 'ptr' is initilized to point to 'a'
    *ptr = 7; // '*ptr' is temporary identifier for 'a'
    printf("%d\n", a); // prints '7'

    ptr = &b; // 'ptr' now points to 'b'
    *ptr = 8; // '*ptr' is temporary identifier for 'b'
    printf("%d\n", b); // prints '8'

    return 0;
}
```

Фрагмент 5: Пример за работа с указатели

Оператор за индирекция

```c
int swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
    printf("%d\n", *a); // prints '5'
    printf("%d\n", *b); // prints '10'
    return 0;
}
```

```c
int main() {
    int a = 10;
    int b = 5;
    swap(&a, &b);
    printf("%d\n", a);// prints '5'
    printf("%d\n", b);// prints '10'
    return 0;
}
```

Фрагмент 6: swap функцията разменя стойностите на променливите, посочени от указателите, които са параметри. Аргументи на функцията са адресите на 'a' и 'b' от main функцията

Оператор за индирекция

- Използването на оператор за индирекция на ненасочен указател, най-често води до грешка и прекратяване на програмата от операционната система

```c
int main() {
    int a = 10;
    int *ptr; /* ptr is not initilized
                 and is pointing to random address */
    *ptr = 10; // crash !
    // printf("%d\n", *ptr); // crashes as well !
    return 0;
}
```

Фрагмент 7: Пример за неправилна работа с указатели

Съдържание

Аритметика с указатели

- В C указателите могат да бъдат събирани с цели числа
- Резултата от събирането на указател с число не е число, а е нов указател
- Новият указател е насочен към адреса на стария указател отместен с толкова байта, колкото е произведението на цялото число с големината в байтове на сочения от стария указател тип
  - Например, ако `ptr` е указател, който сочи към `int` на адрес 1000, а `n` е цяло число, резултата от събирането на `ptr + n` ще е указател, който сочи към адрес: `1000 + (n * sizeof(int))`
- Соченият от новия указател тип е същият като сочения от стария указател

Аритметика с указатели

```c
int main() {
    int a[4] = {1, 2, 3, 4};
    int *ptr = &a[0];
    int *ptr2 = ptr + 2;

    printf("%d\n", *ptr2); // prints '3'
    printf("%d\n", *(ptr + 2)); // prints '3' as well
    printf("%d\n", *(2 + ptr)); // prints '3', addition is commutative

    printf("%d\n", *(ptr2 + (-1)); // prints '2'
    printf("%d\n", *(ptr2 - 1)); // prints '2'
    return 0;
}
```

Фрагмент 8: Събиране и изваждане на указател и число

- Изваждането на цяло число от указател е еквивалентно на събирането на указателя с цялото число с обратен знак

Аритметика с указатели

- Операторите за постфиксно/префиксно инкрементиране/декрементиране могат също да бъдат използвани с указатели

```c
int main() {
    int a[4] = {1, 2, 3, 4};
    int *ptr = &a[0];
    printf("%d\n", *(ptr++)); // prints '1'
    printf("%d\n", *ptr); // prints '2'
    printf("%d\n", *(++ptr)); // prints '3'
    printf("%d\n", *ptr); // prints '3'
    printf("%d\n", *(ptr--)); // prints '3'
    printf("%d\n", *ptr); // prints '2'
    printf("%d\n", *(--ptr)); // prints '1'
    printf("%d\n", *ptr); // prints '1'
    return 0;
}
```

Фрагмент 9: Използване на постфиксни и префиксни ++ и -- с указатели

Аритметика с указатели

- В C е възможно да се изваждат указатели сочещи към един и същ тип
- Резултата от изваждането е цяло число, което оказва разликата между двата указателя в брой променливи от сочения тип

```c
int main() {
    int a[4] = {1, 2, 3, 4};
    int *ptr = &a[0];
    int *ptr2 = &a[2];

    printf("%d\n", ptr2 - ptr); // prints '2'
    printf("%d\n", ptr - ptr2); // prints '-2'
    printf("%d\n", (ptr + 2) - ptr2); // prints '0'
    printf("%d\n", &a[3] - &a[0]); // prints '3'
    return 0;
}
```

Фрагмент 10: Изваждане на указатели

Съдържание

Указатели и масиви

- В C масивите и указателите са тясно свързани
- Идентификаторът (името) на масива може да се използва като указател насочен винаги към първия елемент от масива
  - За разлика от указателите, идентификаторите на масиви не могат да бъдат пренасочвани

```c
int main() {
    int a[4] = {1, 2, 3, 4};
    int *ptr = a; // 'ptr' now points to '&a[0]'

    printf("%d\n", *ptr); // prints '1'
    *(a + 2) = 10; // OK, 'a' is implicitly converted to '&a[0]'
    printf("%d\n", *(a + 2)); // prints '10'
    a++; // ERROR, 'a' is an array, not a pointer
    a = &a[2]; // ERROR, 'a' is an array, not a pointer
    return 0;
}
```

Фрагмент 11: Връзка между указатели и масиви

Указатели и масиви - оператор за индексиране

- Освен с масиви, операторът за индексиране може да се използва и с указатели
- Когато се използва с указател, операторът за индексиране извършва две операции наведнъж - първо събира указателя с подаденото число и след това чрез индирекция върху резултата създава временен идентификатор
- С други думи `p[n]` е по кратък запис на `*(p + n)`

```c
int main() {
    int a[4] = {1, 2, 3, 4};
    int *ptr = &a[0];
    printf("%d\n", *(ptr + 1)); // prints '2'
    printf("%d\n", ptr[1]); // prints '2', as well
    ptr[1] = 9; // OK, 'ptr[1]' is temporary identifier for '&a[1]'
    printf("%d\n", a[1]); // prints '9'
    return 0;
}
```

Фрагмент 12: Употреба на оператор за индексиране с указатели

Указатели и масиви - оператор sizeof

- Важно е да се отбележи, че въпреки привидната заменяемост на име на масив с указател към първия му елемент, има разлики
- Най-важната от тях, е че употребата на оператор sizeof върху указателя, връща броя байтове, който би заела променлива от тип указател, вместо броя в байтове, който заема посоченият масив
- Съответно е невъзможно да се разбере към колко голям масив сочи даден указател

```
int main() {
    int a[4] = {1, 2, 3, 4};
    int *ptr = a;
    printf("%d\n", sizeof(a)); // prints '16' on most computers
    printf("%d\n", sizeof(ptr)); // prints '4' or '8' on most computers
    return 0;
}
```

Фрагмент 13: Разлика при употреба на оператор sizeof с масив и с указател

Обработване на масиви чрез функции

- Обработката на масиви чрез функции може да се извършва само непряко (индиректно)
- Функциите, които обработват масиви, винаги имат параметър от тип указател, който да се насочи към първия елемент от масива за обработка, и в повечето случаи целочислен параметър, който да съдържа броя елементи в масива за обработка
- Имайки тази информация, функцията може успешно да оперира върху масива непряко (индиректно)

```
int square_array(int *ptr, int size) {
    int i = 0;
    for (i = 0; i < size; i++) {
        ptr[i] = ptr[i] * ptr[i];
    }
    return 0;
}
```

```
int main() {
    int a[3] = {1, 2, 3};
    square_array(a, 3);
    return 0;
}
```

Фрагмент 14: Обработване на масиви чрез функции