Структура на програма в C

Част 10 - структури

Иван Георгиев, Христо Иванов, Христо Стефанов

Технологично училище “Електронни системи”,
Технически университет, София

11 юни 2019 г.

Съдържание

Съдържание

Проблем

- При писане на програми често се използват няколко променливи с различни типове за да се опишат някакви логически свързани данни
- Например един от начините за представяне на триъгълник в програма е да се запазят в променливи две от страните му и ъгъла между тях

```
int a = 10;
int b = 5;
double angle = 107.5;
```

Фрагмент 1: Представяне на триъгълник с 3 променливи

Проблем

- При деклариране на още един триъгълник става трудно да се разбере, коя от променливите за кой триъгълник се отнася

```
int a = 10;
int b = 5;
int c = 6;
int d = 8;
double angle = 107.5;
double angle2 = 42.5;
```

Фрагмент 2: Проблем с четене на код

- Чрез избиране на добри имена на променливи, този проблем може да бъде избегнат, но остава проблема, че програмистът трябва допълнително да внимава, когато използва променливите

```
calc_perimeter(tr1_a, tr1_b, tr2_angle); /* tr2_angle not part of tr1 */
```

Фрагмент 3: Подадена грешна променлива при ползване на логически свързани данни

Проблем

- Друг проблем възниква, когато трябва да се декларира “масив от триъгълници”
- За да се “декларира” е нужно да се направят отделни масиви за всяка от променливите, с които се описва триъгълник

```
int a[100];
int b[100];
double angle[100];
```

Фрагмент 4: Декларация на “масив от триъгълници”

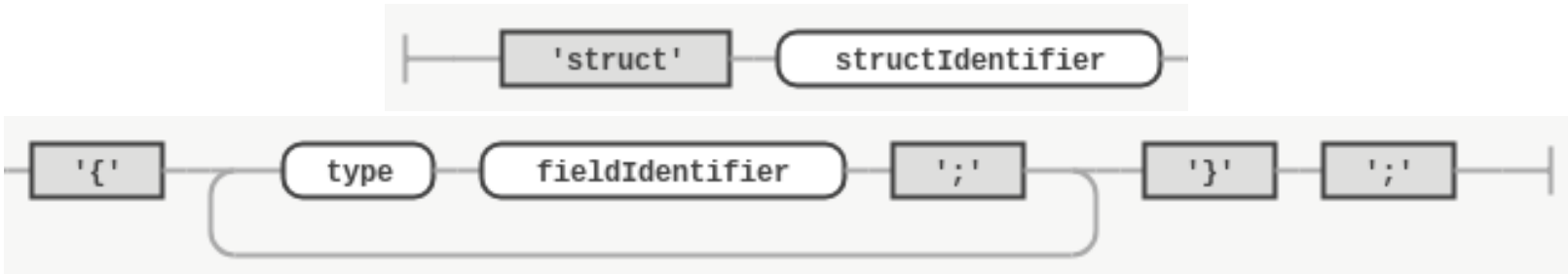
Проблем

- Изброените проблеми дотук са по-скоро неудобства, отколкото ограничения за програмиста
- Въпреки това, поради голямата честота на срещането им, в езика C се предлага механизъм, с който те могат да бъдат избегнати

Съдържание

Структури

- За справянето с тези неудобства, в C е въведена възможността да се създават нови типове данни от програмиста, наречени *структури*
- Структурите позволяват групирането на данни в *полета*, като всяко поле *(field)* може да съдържа различен тип данни
- Полетата в една структура имат имена, които ги различават едно от друго
- Самите структури също имат имена, чрез които се различават една от друга



Фигура 1: Синтактична диаграма на декларация на структура

Структури

```
struct triangle {
    int a;
    int b;
    double angle;
};
```

Фрагмент 5: Пример за декларация на структура описваща триъгълник

```
struct circle {
    double x;
    double y;
    double radius;
};
```

Фрагмент 6: Пример за декларация на структура описваща окръжност в координатна система

Структури

- Декларираните структури могат да бъдат използвани в една програма, навсякъде където се очаква тип данни
- Начинът, по който това става, е чрез специален синтаксис - на мястото в програмата, където се очаква тип, се поставя ключовата дума `struct`, следвана от името на структурата

```
struct triangle a; /* variable of type 'struct triangle' */
struct triangle b; /* variable of type 'struct triangle' */
struct triangle triangles[10]; /* array with elements of type
                                  'struct triangle' */
struct triangle* p = &a; /* pointer to 'struct triangle' variable */
struct triangle func1(struct triangle t); /*
    function returning a 'struct triangle' with one parameter
    of type 'struct triangle'
*/
```

Фрагмент 7: Примери за декларации използващи структурата `triangle`

Инициализиране на променливи от структурен тип

- Променливите от структурен тип, могат да бъдат инициализирани, чрез инициализиращ списък
- Поредната стойност в инициализиращия списък се използва за начална стойност на поредното поле от структурата
- Типът на стойността трябва да отговаря на типа на полето, което инициализира
- Ако инициализиращият списък е изпуснат, полетата на структурата имат произволни стойности
- Ако не са инициализирани всички полета, неинициализираните полета приемат за начална стойност 0

```
struct triangle a = {3, 4, 90.0}; // OK
struct triangle b = {3, 4}; // OK, angle is set to 0.0
```

Фрагмент 8: Инициализиране на променливи от тип структура triangle

Съдържание

Оператор за достъпване на поле

- Полетата на структурата на променлива от структурен тип могат да бъдат достъпвани чрез *оператора за достъпване на поле от структура*

Фигура 2: Синтактична диаграма на оператор за достъпване на поле от структура

- За първи операнд на оператора се подава име на променлива, а за втори - име на поле от структурата, от която е променливата
- Достъпеното поле се държи като променлива, тъй като резултата от прилагането на оператора е `lvalue`

Оператор за достъпване на поле

```c
struct triangle t = {3, 4, 90.0};
t.angle = 47.5;
printf("%f\n", t.angle);
```

Фрагмент 9: Употреба на оператор за достъпване на поле от структура

```c
struct triangle t = {3, 4, 90.0};
t.side_a = 47.5; /* ERROR, 'side_a' is not a field
                    in 'struct triangle' */
printf("%f\n", t.side_a); /* ERROR, same as above */

int q = 10;
q.side_a = 4; /* ERROR, 'q' does not have a struct type */
```

Фрагмент 10: Грешна употреба на оператор за достъпване на поле от структура

Оператор за присвояване

- За разлика от масивите, променливите от структурен тип могат да бъдат присвоявани една на друга
- При присвояване, стойностите на полетата от едната променлива се присвояват на полетата в другата променлива
- Задължително условие е, структурните типове на двете променливи да съвпадат

```
struct triangle t = {3, 4, 90.0};
struct triangle t2 = {10, 20, 17.5};
t2 = t;
printf("%f\n", t2.angle); /* prints '90.0' */
```

Фрагмент 11: Присвояване на променливи от структурен тип

Оператор за присвояване

```c
struct rectangle {
    int a;
    int b;
};

struct triangle t = {3, 4, 90.0};
struct rectangle r = {10, 20};
t = r; /* ERROR, 't' is of type 'struct triangle' and
                  'r' is of type 'struct rectangle' */
```

Фрагмент 12: Грешна употреба на оператор за присвояване

Указатели към променливи от структурен тип

- Подобно на променливите от основен тип, указатели могат да се насочват и към променливи от структурен тип
- Указателите насочени към променливи от структурен тип, се държат по същия начин както и указателите насочени към основен тип

```
struct triangle t = {3, 4, 90.0};
struct triangle *p = &t;
(*p).a = 10;
printf("%d\n", (*p).a); /* prints '10' */
```

Фрагмент 13: Пример с указател насочен към променлива от структурен тип

- Скобите в израза `(*p).a`, са нужни, тъй като оператора за достъп до поле на структура има по-висок приоритет от оператора за индирекция и би се получила компилационна грешка

Указатели към променливи от структурен тип

- Тъй като доста често се налага да се използва оператора за индирекция последван от оператора за достъп до поле на структура, в C има оператор, който комбинира двете операции в една - *оператор за достъп до поле на структура на посочена променлива от структурен тип*
- Този оператор се обозначава със стрелка '->' (тире последвано от знак за по-голямо) и е по-известен под това име - *оператор стрелка*

Фигура 3: Синтактична диаграма на оператор стрелка

- Първият операнд е име на указател към променлива от структурен тип, а вторият е име на поле от структурата на променливата

Указатели към променливи от структурен тип

```c
struct triangle t = {3, 4, 90.0};
struct triangle *p = &t;

/* the following two fragments are equivalent */

(*p).a = 20;
printf("%d\n", (*p).a); /* prints '20' */

p->a = 20;
printf("%d\n", p->a); /* prints '20', as well */
```

Фрагмент 14: Пример с оператор стрелка

Указатели към променливи от структурен тип

```c
struct triangle t = {3, 4, 90.0};
struct triangle *p = &t;

(*p).a = 3;
p->b = 4;
p->angle = 17.5;

printf("%d\n", p->a); /* prints '3' */
printf("%d\n", (*p).b); /* prints '4' */
printf("%d\n", t.angle); /* prints '17.5' */
```

Фрагмент 15: Пример с оператор стрелка

Съдържание

Разположение в паметта на структури

- Полетата на една структура винаги се разполагат последователно в паметта на компютъра
- За разлика от масивите, между отделните полета може да има допълнителни празни байтове (наречени *байтове за подравняване, padding*)
- Дали да има или да няма байтове за подравняване е решение на компилатора
- Поради тази причина големината в байтове на една структура може да е по-голяма от сбора на големините на отделните полета
- За да се разбере, колко байта заема една структура в паметта, трябва да се използва оператор `sizeof`

Разположение в паметта на структури. оператор sizeof

```c
struct side {
    char name;
    double length;
};
struct point {
    int x;
    int y;
};
int main() {
    /* the exact sizes depend on the target processor. mine are: */
    printf("%zd\n", sizeof(int)); /* prints 4 */
    printf("%zd\n", sizeof(struct point)); /* prints 8 */
    printf("%zd\n", sizeof(char)); /* prints 1 */
    printf("%zd\n", sizeof(double)); /* prints 8 */
    printf("%zd\n", sizeof(struct side)); /* prints 16 */
    return 0;
}
```

Фрагмент 16: Разлика между големина на полета и големина на структура