Лекция 12. Структуры и другие формы данных. Часть 1.

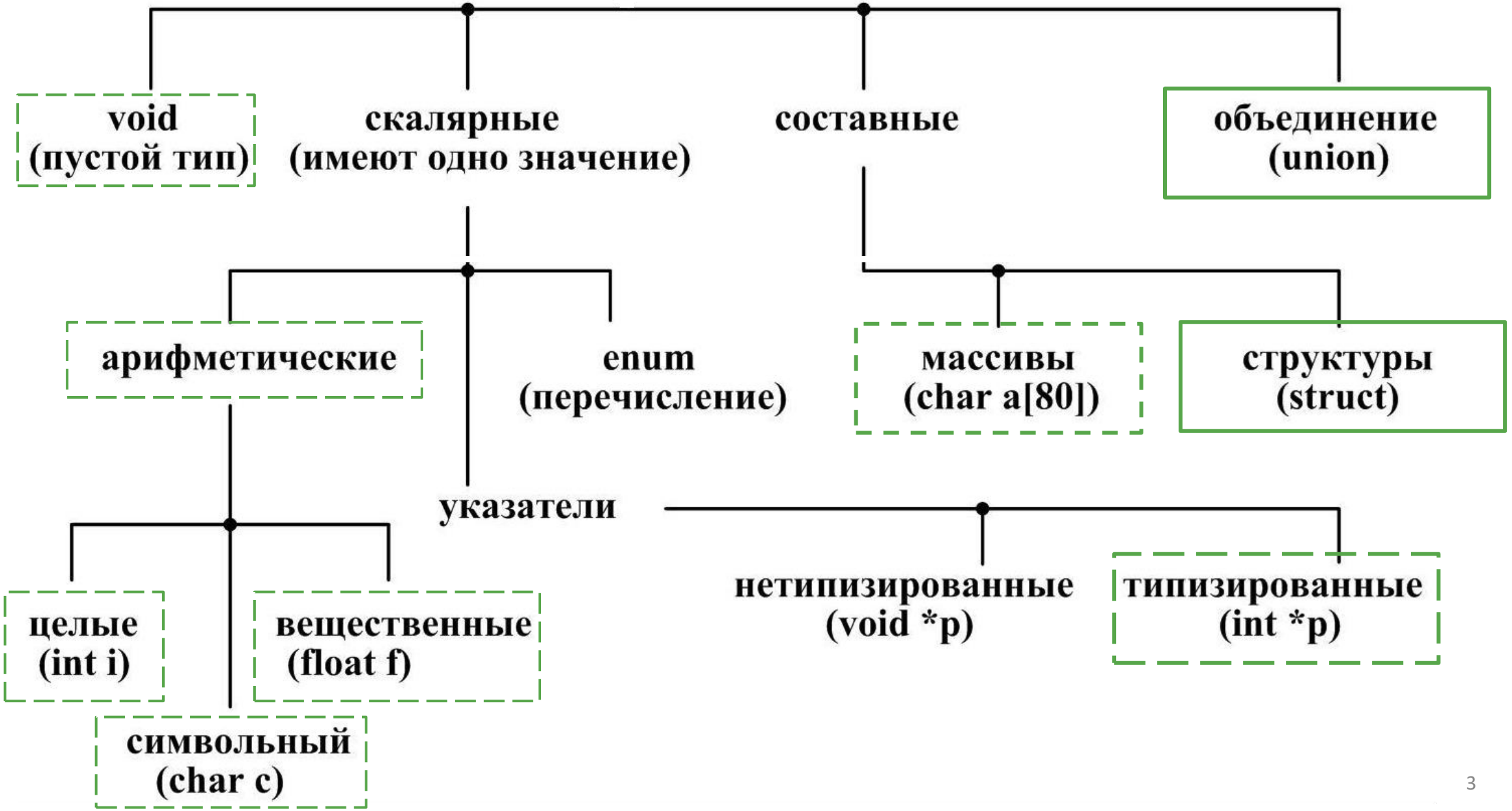
Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Содержание

- Ключевые слова: struc.
- Операции: . ->
- Структуры в языке Си и способы создания шаблонов и переменных типа структур
- Доступ к членам структуры и написание функций для обработки структур

Типы данных в Си (прогресс)



Новые подходы в представлении данных

Представление данных один из важнейших этапов при разработке программы.

Во многих случаях простой *переменной* или даже *массива* оказывается недостаточно. Язык **Си** позволяет расширить возможности *представления данных* с помощью *переменных* типа **структур**.

В своей базовой форме **структура Си** является достаточно гибким средством, чтобы представлять широкое разнообразие данных, и она позволяет изобретать новые формы *представления данных*.

Объявления структуры (1)

Создание структуры предполагает описание “схемы”, которая формирует представление структуры в памяти.

```
struct book {
    char title[MAXTITL]; // #define MATITL 41
    char author[MAXAUTL]; // #define MAXAUTL 31
    float value;
);
```

Данное описание не создаёт ***объект данных,*** а лишь определяет из чего он состоит. (образованную из двух символьных массивов и одной переменной типа `float`).

Объявления структуры (2)

```
struct book {
  char title[MAXTITL];//#define MATITL 41
  char author[MAXAUTL];//#define MAXAUTL 31
  float value;
);
```

Первым идет ключевое слово `struct`. Оно указывает, что за ним следует структура. Далее находится необязательный ***дескриптор*** — слово book — сокращенная метка, которую можно применять для ссылки на эту *структуру*.

Объявления структуры (3)

```
struct book {
  char title[MAXTITL];//#define MATITL 41
  char author[MAXAUTL];//#define MAXAUTL 31
  float value;
};
```

Далее внутри **блочного оператора** следует перечисление содержащихся в нём значений, которые называют - *члены структур*. Точка с запятой после закрывающей фигурной скобки завершает определение шаблона структуры.

Объявления структуры (4)

```
struct book {
  char title[MAXTITL];//#define MATITL 41
  char author[MAXAUTL];//#define MAXAUTL 31
  float value;
};
```

Если структура описана за пределами функции (в заголовочном файле, после указания библиотек и т.п.) – она доступна глобально. Если она описана внутри функции, тогда только применима в пределах **блочного оператора** объявления.

Создание переменной из структуры (1)

```
struct book library;
```

Объявляет переменную `library` типом данных *структуры* `book`. Раннее определение шаблона `book` позволит компилятору выделить память в соответствии с описанными членами структуры.

Имя		**struct book** library char * title	char * author	float value	-	-
Адрес	0x06	0x07	0x48	0x79	0x80	0x81
Значение	й	B	T	№	a	0
Индекс		0	0			
		title[0]	author[0]			

```
struct book prog, * design;
```

Создание переменной из структуры (2)

```
struct book {
  char title[MAXTITL];//#define MATITL 41
  char author[MAXAUTL];//#define MAXAUTL 31
  float value;
} library;
```

Имя		**struct book** library char * title	char * author	float value	-	-
Адрес	0x06	0x07	0x48	0x79	0x80	0x81
Значение	й	B	T	№	a	0
Индекс		0	0			
		title[0]	author[0]			

Однако, если структура используется более одного раза лучше задавать ***дескриптор***.

Инициализация структуры (1)

```
struct book library = {"Язык
программирования Си. 6-е издание",
"Стивен Прата", 2245.00};
```

Для инициализации *структуры* заполняется список разделяемый запятыми, заключенный в фигурные скобки. Тип каждого инициализатора должен соответствовать *типу члена структуры*.

Доступ к членам структуры (1)

```
struct book library = {"Язык программирования Си. 6-е издание", "Стивен Прата", 2245.00};
```

Для получения доступа к членам структуры используется **оператор членства в структуре** – точка (.)

```
strncpy(library.author,"Толстой", MAXAUTL);
strncpy(library.title,"Война и Мир", MAXTITL);
library.value = 1842.00;
```

Назначенные инициализаторы(1)

Назначенные инициализаторы позволяют установить значение членов структуры используя **оператор (.)** и название члена структуры.

```
struct book surprise = {.value = 10.99};
struct book gift1 = {.value = 25.99,
                     .author = "James Broadfool",
                    .title = "Rue for the Toad"};
struct book gift2 = {.value = 18.90,
                    .author = "Philionna Pestle",
                   0.25};
```

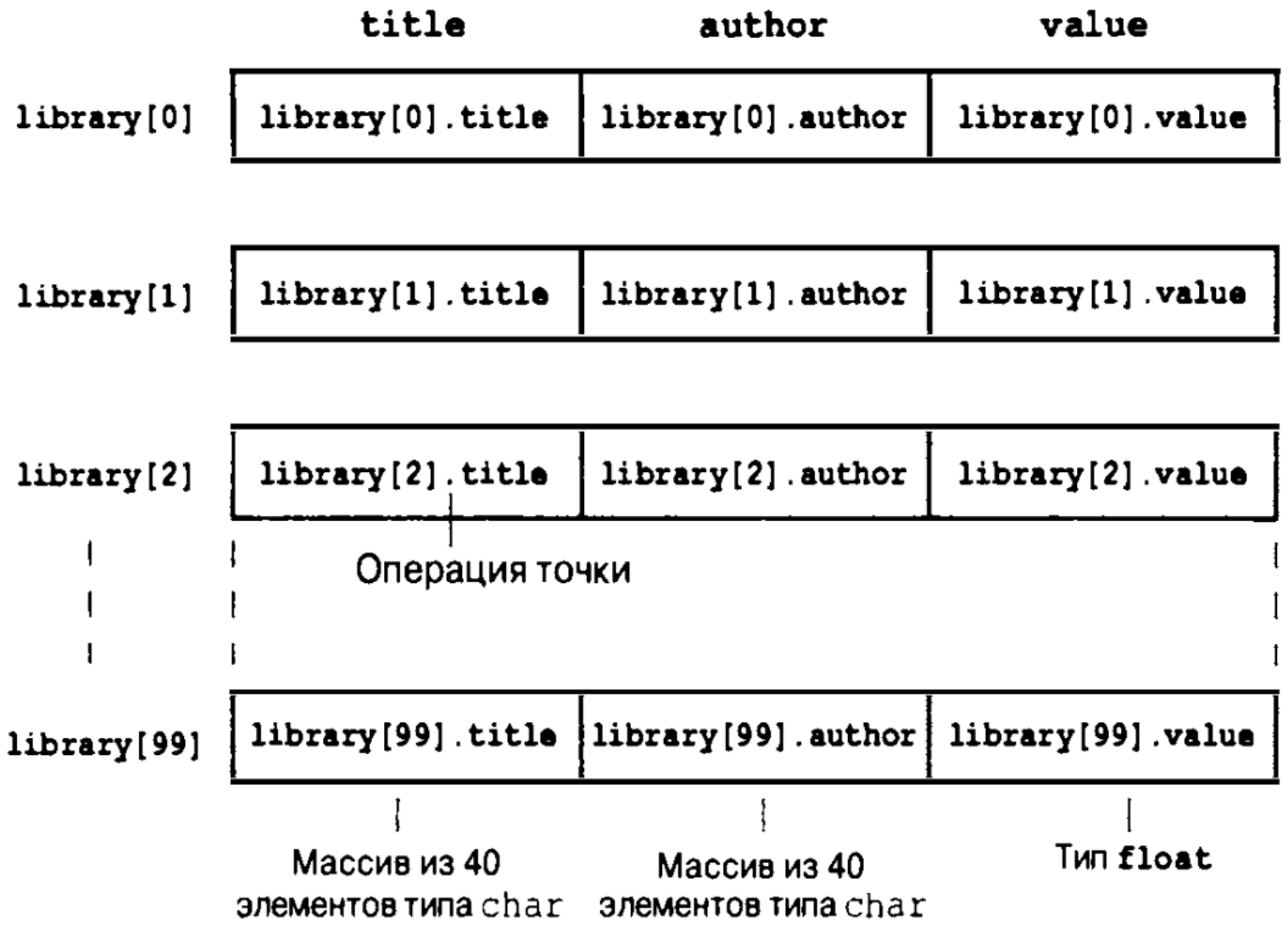
Массивы структур (1)

Язык Си позволяет создавать *массивы структур*.

```
#define MAXBKS 10
struct book library[MAXBKS];
```

Каждый *элемент массива* в данном случае трактуется как отдельная *структура* типа `book`.

Массивы структур (2)



Объявление: `struct book library[MAXBKS]`

Массивы структур (3)

При использовании *массивов* для идентификации членов в *массиве структур* применяются те же самые правила, что и в случае индивидуальных структур: за именем структуры должна следовать операция точки, а затем имя члена.

```
library[0].author
library[0].title
library[0].value
~~library.title[0]~~
```

Массивы структур (4)

```
struct book library[10] = {0};
strncpy(library[0].author,"Фёдор
Достоевский", 31);
strncpy(library[0].title,"Бесы", 41);
printf("library[0]\n .author = %s\n
.title = %s\n .value = %f\n",
library[0].author, library[0].title,
library[0].value);
```

Вложенные структуры (1)

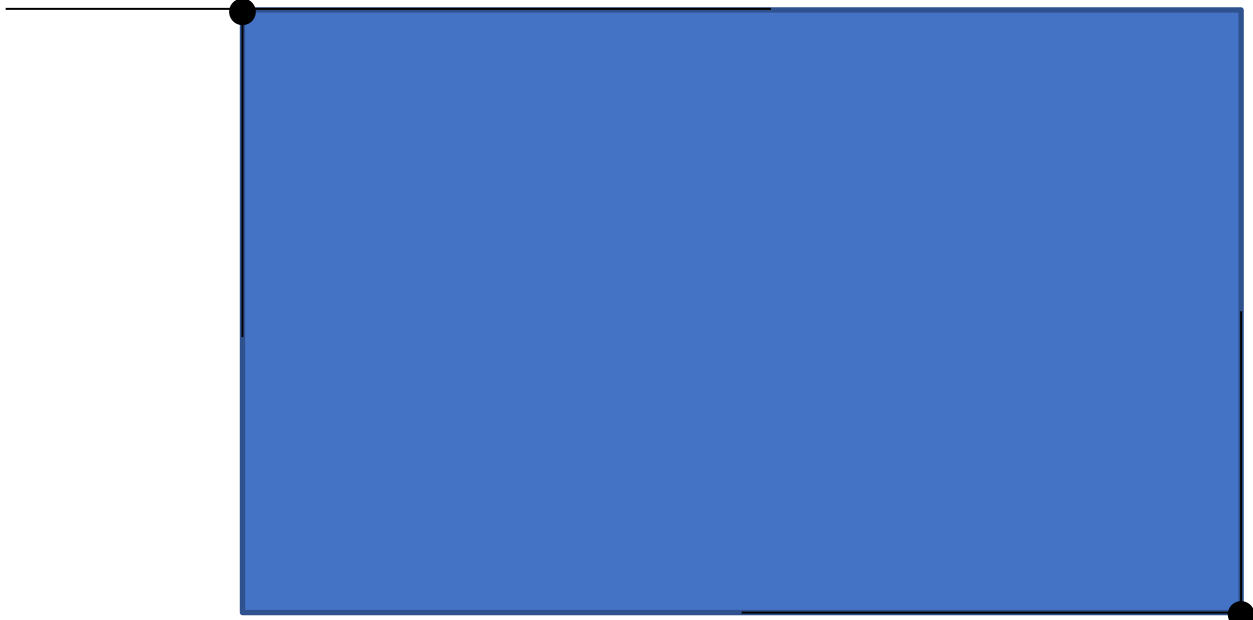
Язык Си позволяет вкладывать *структуры* в другие *структуры* по аналогии с многомерными массивами. Структура входящая в описание другой *структуры*, называется *вложенной*.

```
struct coord {
    int x,y;
};
```

```
struct rect {
    struct coord
left_corner;
    struct coord
right_corner;
}
```

Вложенные структуры (2)

```
rect.left_corner = {coord.x,
                    coord.y}
```



```
rect.right_corner = {coord.x,
                     coord.y}
```

Вложенные структуры (3)

Для обращения к вложенной структуре используется **оператор членства в структуре.**

```
struct rect box;
        box.left_corner.x = 10;
        box.left_corner.y = 10;
        box.right_corner.x = 20;
        box.right_corner.y = 20;
```

Указатели на структуры (1)

Указатели на *структуру* предоставляют ряд преимуществ:

- Манипулировать указателями проще.
- *Структура* не может быть передана как аргумент функции (уст.).
- Передача указателя эффективнее.
- *Структуры* могут содержать *указатели* на *структуры*.

Указатели на структуры (2)

```
struct rect boxes[2] =
{{{10,10},{20,20}},{{40,40},{60,60}}};
struct rect * box1;
printf("адрес #1: %p #2: %p \n",
&boxes[0], &boxes[1]);
box1 = &boxes[0];
printf("указатель #1: %p #2: %p \n ",
box1, box1 + 1);
```

Указатели на структуры (3)

```
printf("box->left_corner->x равно %d:
(*(*box).left_corner).x равно %d\n",
box1->left_corner.x,
(*box1).left_corner.x); box1++;
printf("box->left_corner->x равно %d:
(*(*box).left_corner).x равно %d\n",
box1->left_corner.x,
(*box1).left_corner.x);
```

Указатели на структуры (4)

```
адрес #1: 0000003B3D9FF780
 адрес #2: 0000003B3D9FF790
указатель #1: 0000003B3D9FF780
указатель #2: 0000003B3D9FF790
 box->left_corner->x равно 10
(*(*box).left_corner).x равно 10
box->left_corner->x равно 40
(*(*box).left_corner).x равно 40
```

Указатели на структуры (5)

В отличии от массивов, имя *структуры* не является ее адресом, для получения адреса структуры нужно применять **операцию разыменовывания &**.

```
struct rect * box2;
box2 = &boxes[1];
```

Указатели на структуры (6)

`box2 = &boxes[1];`

`box2` – не является именем структуры, поэтому нет возможности получить значение члена от указателя используя **операцию точки(.).** Для этого потребуется следующая **операция** `->` (дефис и символ больше).

`box2->left_corner.x == boxes[1].left_corner.x`

если `box2==&boxes[1];` Однако есть второй способ:

`box2 = &boxes[1];` тогда `*box2 = boxes[1];`

следовательно `boxes[1].left_corner.x == (*box2).left_corner.x;`

Функции и структуры (1)

Передача значения члена структуры в функцию, возможно при условии если он имеет тип данных где единственным значением (т.е. **int** или один из его производных типов, **char**, **float**, **double** либо **указатель**), его можно передавать в качестве аргумента функции, которая принимает этот конкретный тип.

Функции и структуры (2)

```
int sumint(int x, int y) return (x+y);
struct coord { int x, y; };
…
int main(){
struct coord points[2] = {{2,2},{5,5}};
sumint(points[0].x, points[1].x);
… }
```

Функции и структуры (3)

По аналогии, данную задачу можно решить передавая в качестве аргументов адрес структуры. Однако имя структуры не является ее адресом, и для получения адреса структуры обязательно следует использовать **операцию разыменовывания &**.

Функции и структуры (4)

```
int sumint(const struct coord * p
)
    {return (p->x+p->y);}
struct coord { int x, y; };
…
int main(){
struct coord points[2] = {{2,2},{5,5}};
sumint(&points[0]);
… }
```

Функции и структуры (5)

Некоторые компиляторы (не все) разрешают передавать *структуры* в качестве аргументов. В таком случае функция создаст *копию структуры* во время выполнения функции, после выполнения функции *копия будет удалена* и это никаким образом не сможет оказать влияние на *оригинал структуры*. В отличи от предыдущего примера с указателем на *структуру*.

Функции и структуры (6)

```
int sumint(const struct coord p
)
    {return (p.x+p.y);}
struct coord { int x, y; };

…
int main(){
struct coord points[2] = {{2,2},{5,5}};
sumint(points[0]);
… }
```

Особенности структур (1)

В отличии от *массивов* при работе со *структурами*, можно использовать **операцию присваивания =**, при условии что *объекты данных* имеют один и тот же *тип данных*.

```
struct coord point1 = {2,2};
struct coord copy_point1 = point1;
printf("x=%d y=%d",copy_point1.x,
copy_point1.y);
```

Особенности структур (2)

Структуры можно возвращать из функции при помощи **оператора return**. Использование функций для возвращения *структур* дает возможность передавать информацию о *структуре* из вызываемой функции в вызывающую . *Указатели на структуры* также допускают двусторонний обмен данными, так что вы часто будете применять один из этих подходов при решении разнообразных задач.

Особенности структур (3)

```
struct coord {
    int x,y;
};
struct coord setPoint(void);
double rangeTwoPoint(const struct
coord f, const struct coord s);
void showPoint(const struct coord c);
```

Особенности структур (4)

```
struct coord setPoint(void){
    int x=0,y=0;
    struct coord buff= {};
    printf("Введите координаты точки:");
    scanf("%d %d",&x,&y);
    buff.x = x; buff.y = y;
    return buff;
};
```

Особенности структур (5)

```
void showPoint(const struct coord c)
{
    printf("Точка имеет коодинаты x=%d,
y=%d\n", c.x,c.y);
}
```

Особенности структур (6)

```
double rangeTwoPoint(const struct coord
f, const struct coord s)
{
    double r=sqrt(pow(f.x-s.x,2) +
                  pow(f.y-s.y,2));
     return r;
}
```

Особенности структур (7)

```
struct coord point1, point2, point3;
point1 = setPoint();
showPoint(point1);
point2 = setPoint();
showPoint(point2);
printf("Расстояние между двумя точками =
%lf",rangeTwoPoint(point1, point2));
```

Особенности структур (8)

```
Введите координаты точки:10 10
Точка имеет коодинаты x=10, y=10
Введите координаты точки:5 5
Точка имеет коодинаты x=5, y=5
Расстояние между двумя точками = 7.071068
```

Особенности структур (8)

Представленный подход применим только к *небольшим структурам*, которые не предполагают хранение **больших объёмов данных**. В противном случае получится очень **медленно** и **“тяжело”** по памяти если передавать целые *структуры*. Более эффективным подходом, является использование *указателей*.

Строки, указатели char и структуры (1)

```
#define LEN 20
struct names {
    char first[LEN];
    char last[LEN];
};
struct pnames {
    char * first;
    char * last;
};
```

Строки, указатели char и структуры (2)

```
    struct names
talia = {"Talia", "Summers"};

    struct pnames
brad = ("Brad", "Fallingjaw"};
```

Строки, указатели char и структуры (3)

```
struct names accountant;
struct pnames attorney;
puts ("Введите фамилию вашего
бухгалтера:");
scanf("%s", accountant.last);
puts ("Введите фамилию вашего
адвоката:");
scanf("%s", attorney.last) ;
```

Составные литералы и структуры (1)

Средство составных литералов С 99 доступно для структур), а также для массивов.

```
struct book {
      char title [41] ;
      char author [31] ;
      float value;
};
(struct book) {"Идиот", "Федор
Достоевский", 6.99}
```

Составные литералы и структуры (2)

Составные литералы можно использовать также как аргументы функций. Если функция ожидает *структуру*, то ей можно передавать в качестве фактического аргумента составной литерал.

```
struct rect {double x; double y;};
double rect_area(struct rect r)
    {return r.x * r.y;}
double area;
area = rect_area((struct rect)
{10.5, 20.0));
```

Составные литералы и структуры (2)

Составные литералы можно использовать также как аргументы функций. Если функция ожидает структуру, то ей можно передавать в качестве фактического аргумента составной литерал.

```
struct rect {double x; double y;};
double rect_area(struct rect r)
    {return r.x * r.y;}
double area;
area = rect_area((struct rect)
{10.5, 20.0));
```

Анонимные структуры (1)

Стандарт C11 вводит новое понятие, *анонимная структура*. Другая именованная *структура*, может содержать в себе не именованную (**анонимную**) *структуру*, как член именованной структуры. Применение этого подхода позволяет избежать создания *структур* которые используются исключительно *для вложения в* другую *структуру*, но при этом **упрощает обращение, инициализацию, обработку**.

Анонимные структуры (2)

```
struct names{
     char first[20] ;
     char last[20];} ;
struct person{
     int id;
     struct names name; } ;
struct person ted = {8483,
        {"Ted", "Grass"}};
puts(ted.name.first);
```

Анонимные структуры (3)

```
struct person{
     int id;
     struct {char first[20];
       char last[20];};};
};
struct person ted = {8483, {"Ted",
"Grass"}};
puts(ted.first);
```

Задачи на практику

Читать:

1. Глава 14 Структуры и другие формы данных(стр. 565-587, 591-595).

Выполнить **ПОСЛЕДНЮЮ** практику по желанию:

1. https://eios.sibsutis.ru/mod/resource/view.php?id=168180

Спасибо за внимание

Ревун Артем Леонидович
ст. преп. Кафедры вычислительных систем

Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)