Лекция 9. Массивы и указатели. Часть 2.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

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

Содержание лекции

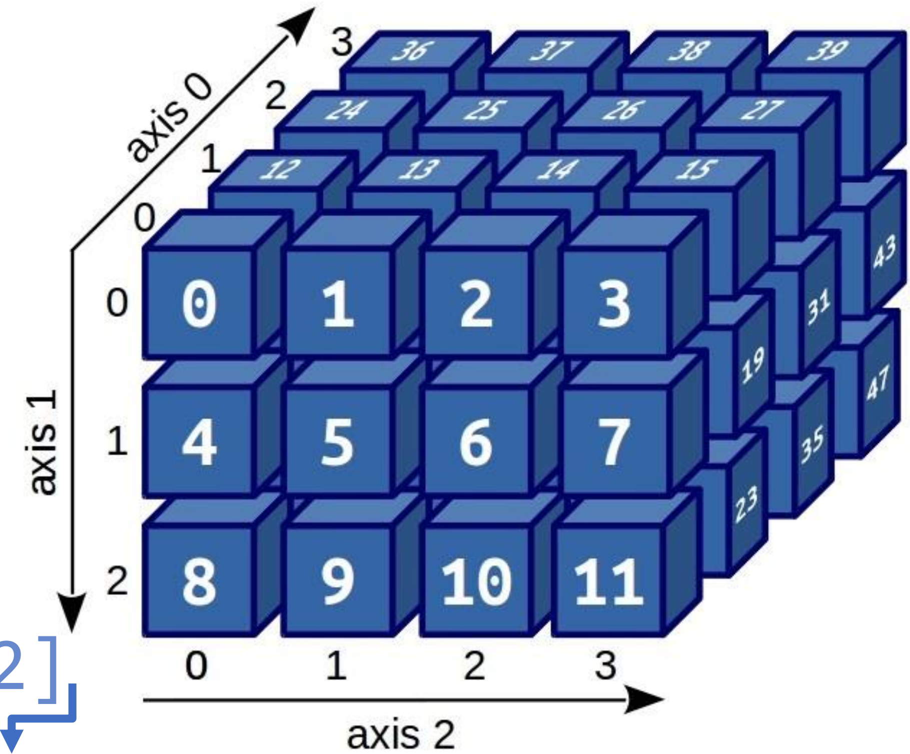
- Ключевое слово: **static**
- Операции: & * (унарная)
- Указатели (на основе сведений, которые вам уже известны) и их отношение к массивам
- Многомерные массивы и обработка содержащейся в них информации

Много многомерности

В языке Си нет ограничения касательно представления уровня вложенности при использовании массивов.

```
int cube[4][3][4];
```

Трактуется как создание четырех двухмерных массивов в три строки по четыре элемента int. Абстракция сложна для представления, однако представление в памяти не изменилось.



cube[0][0] cube[0][1] cube[0][2]

Имя	-	cube	-	-	-	-	-	-	-	-	-	-	-	-	-
Адрес	0x04	0x08	0x0C	0x10	0x14	0x18	0x1C	0x20	0x24	0x28	0x2C	0x30	0x34	0x38	0x3C
Значение	2453	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Индекс	-	0	1	2	3	0	1	2	3	0	1	2	3	0	1

cube[0][0][0] cube[1][0][0]

Указатели на массивы

Аппаратные инструкции вычислительных машин полагаются на **адреса**, отсюда следует что программы с указателями эффективнее. **Указатели** в свою очередь лежат в основе манипулирования массивами. Следует разобраться!

```
int arr[5]  = {1,2,3,4,5};
printf("arr=%p - &arr[0]=%p", arr, &arr[0]);
arr=000000d4b67ff780 - &arr[0]=000000d4b67ff780
```

`arr, &arr[0]` - являются константами, т.к. адреса фиксированы в выделенном адресном пространстве программы. Тем не менее указатель можно присваивать как значение переменной.

Простая арифметика указателей (1)

```
short sarr[] = {2,4,8,16,32};
double darr[] = {2.,4.,8.,16.,32.};
printf("Стартовые адреса short=%p double=%p\n", sarr,
darr);
printf("Адрес+1 short=%p double=%p\n", sarr+1, darr+1);
printf("Адрес+2 short=%p double=%p\n", sarr+2, darr+2);
printf("Адрес+3 short=%p double=%p\n", sarr+3, darr+3);
printf("Адрес+4 short=%p double=%p\n", sarr+4, darr+4);
```

```
Стартовые адреса short=000000557dfffb06 double=000000557dfffad0
Значение + 1 адреса short=000000557dfffb08 double=000000557dfffad8
Значение + 2 адреса short=000000557dfffb0a double=000000557dfffae0
Значение + 3 адреса short=000000557dfffb0c double=000000557dfffae8
Значение + 4 адреса short=000000557dfffb0e double=000000557dfffaf0
```

Простая арифметика указателей (2)

```
short sarr[] = {2,4,8,16,32};
&sarr[0]=000000557dfffb06
&sarr[1]=000000557dfffb08
&sarr[2]=000000557dfffb0a

000000557dfffb06 + 1 = 000000557dfffb08
000000557dfffb06 + 1*(short) => 06+02(byte) = 08
000000557dfffb08 + 1 = 000000557dfffb0a
000000557dfffb08 + 1*(short) => 08+02(byte) = 10
000000557dfffb06 + 2 = 000000557dfffb0a
000000557dfffb06 + 2*(short) => 06+04(byte) = 10
```

Простая арифметика указателей (3)

```
double darr[] = {2.,4.,8.,16.,32.};
&darr[0]=000000557dfffad0
&darr[1]=000000557dfffad8
&darr[2]=000000557dfffae0

000000557dfffad0+1=000000557dfffad8
000000557dfffad0+1*(double)=>d0+08(byte)=d8
000000557dfffad8+1=000000557dfffae0
000000557dfffad8+1*(double)=>d8+08(byte)=e0
000000557dfffad0+2=000000557dfffae0
000000557dfffad0+2*(double)=>d0+10(byte)=e0
```

Простая арифметика указателей (4)

Приведенные примеры показывают принцип адресной арифметики.

В контексте *указателей* операция суммы означает добавление одной *единицы хранения*.

В контексте *массива*, *указатель* изменяется до адреса следующего элемента.

Размер единицы хранения зависит от типа данных на который ссылается *указатель*. <u>*(Для этого указателю и нужен тип данных).*</u>

Функции, массивы и указатели (1)

```
int sum(int * array);
int main(){
    int marbles[] = {1,2,3,4,5};
    int total = sum(marbles);
}
int sum(int * array)
{
int total = 0;
for (int i = 0; i < 10; i++)
    total += array[i];
return total;
}
```

Функции, массивы и указатели (2)

```
int sum(int * array, int n);
int main(){
    int marbles[] = {1,2,3,4,5};
    int total = sum(marbles);
}
int sum(int * array, int n)
{
int total = 0;
for (int i = 0; i < n; i++)
    total += array[i];
return total;
}
```

Функции, массивы и указатели (3)

```
int sum(int *array, int n); //вариант 1
int sum(int *, int); //вариант 2
int sum(int array[], int n); //вариант 3
int sum(int [], int); //вариант 4
…
int sum(int *ar, int n){/*код*/} //вариант 1
int sum(int ar[], int n){/*код*/} //вариант 2
```

Поскольку имя массива — это адрес его первого элемента, фактический аргумент в виде имени массива требует, чтобы соответствующий формальный аргумент был указателем. В этом и только в этом контексте С интерпретирует int ar [ ] как int * ar, т.е. ar является типом указателя на int.

Функции, массивы и указатели (4)

При передаче массива в качестве аргумента функции, требуется передать его *тип*, *адрес* и *размер*. Альтернативный подход предполагает передачу начала и конца массива.

```
int sumptr(int *arrst, int *arrend);
int main(){
    int arr[SIZE] = {1,2,4,8,16};
    printf("%d", sumptr(arr, arr+SIZE));}
int sumptr(int *arrst, int *arrend){
    int total = 0;
    while (arrst < arrend){
    total += *arrst++; //* и ++
    }return total;}
```

Операции с указателями (1)

Присваивание. Указателю можно присвоить адрес. Присваиваемым значением может быть: имя массива[1], переменная, которой предшествует операция взятия адреса (&) [2], или другой указатель [3].

[1]
```
int arr[SIZE] = {1,2,4,8,16}; int *ptr = arr;
```
[2]
```
int number = 1; int *ptr2 = &number;
```
[3]
```
int *ptr3 = NULL;int *ptr4 = ptr3;
```

Адрес должен быть совместим с типом указателя.

```
double d=1.; double *ptrd=&d; int *ptri=ptrd;
double d=1.; double *ptrd=&d;
int *ptri=(int *)ptrd; //Без явного преобразования
эта операция приведёт к предупреждению.
```

Операции с указателями (2)

Нахождение значения (разыменование). Операция * дает значение, хранящееся в ячейке, на которую указывает указатель.

```
double d  = 1.;
printf("%lf - %p", d, &d);
1.000000 - 0000003cad9ffcc0
```

Операции с указателями (3)

Взятие адреса указателя. Подобно *переменным*, переменная типа *указателя* имеет *адрес* и *значение*. **Операция &** сообщает, где хранится сам указатель.

```
int arri[3] = {1,2,3};
int *ptri = arri;
printf("value=%p, adress=%p", ptri, &ptri);
```

Операции с указателями (4)

Добавление целого числа к указателю. С помощью **операции** + можно добавить целое число к указателю. Целое число умножается на количество байтов в *типе данных*, на который указывает *указатель*, и результат добавляется к исходному *адресу*.

```
int arri[3] = {1,2,3}; int *ptri = arri;
printf("%p->%p->%p->%p\n",ptri,ptri+1,ptri+2,ptri+3);
printf("%d->%d->%d->%d\n",*(ptri),*(ptri+1), *(ptri+2),
*(ptri+3));
printf("%p  -  %d\n", ptri+1000, *(ptri+1000));
```

Результат сложения не определен, если он находится за пределами массива, на который указывает исходный указатель.

Исключением будет адрес, следующий за последним элементом массива.

```
000000FF3C1FFE4C -> 000000FF3C1FFE50 -> 000000FF3C1FFE54 -> 000000FF3C1FFE58
1 -> 2 -> 3 -> 1008729676
```

Операции с указателями (5)

Инкрементирование указателя. Приводит к его перемещению на следующий элемент массива.

```
int arri[3] = {4,8,16}; int *ptri = arri; //arr[0] 0x08
arri++; //arri[1] 0x08+4byte=0x0C
arri++; //arri[2] 0x0C+4byte=0x10
```

Имя	*ptri	arri[0]	arri[1]	arri[2]	-	-
Адрес	0x04	0x08	0x0C	0x10	0x14	0x18
Значение	0x08	4	8	16	38283	9862
Индекс	-	0	1	2	-	-

Переменная не перемещается в памяти лишь потому, что изменилось ее значение!

Операции с указателями (6)

Вычитание целого числа к указателю. С помощью **операции -** можно вычесть целое число из указателя. *Указатель* должен быть первым операндом, а *число* вторым! Целое число умножается на количество байтов в *типе данных*, на который указывает *указатель*, и результат вычитается из исходного *адреса*.

```
int arri[3] = {1,2,3}; int *ptri = arri+3;
printf("%p->%p->%p->%p\n",ptri,ptri-1,ptri-2,ptri-3);
printf("%d->%d->%d->%d\n",*(ptri),*(ptri-1),*(ptri-2),
*(ptri+3));
printf("%p  -  %d\n", ptri-1000, *(ptri-1000));
```

Результат вычитания не определен, если он находится за пределами массива, на который указывает исходный указатель.

Исключением будет адрес, следующий за последним элементом массива.

Операции с указателями (7)

Декрементирование указателя. Приводит к его перемещению на предидущий элемент массива.

```
int arri[3] = {4,8,16}; int *ptri = arri+3; //arr[0] 0x08
arri--; //arri[1] 0x10-4byte=0x0C
arri--; //arri[0] 0x0C-4byte=0x08
```

Имя	*ptri	arri[0]	arri[1]	arri[2]	-	-
Адрес	0x04	0x08	0x0C	0x10	0x14	0x18
Значение	0x10	4	8	16	38283	9862
Индекс	-	0	1	2	-	-

Использование операций инкремента и декремента можно использовать и префиксную форму и постфиксную форму.

Операции с указателями (7)

Разность. Можно находить разность между двумя указателями. Обычно это делается для двух *указателей* на элементы, находящиеся в одном массиве, чтобы определить, насколько далеко они отстоят друг от друга. Результат представлен в тех же единицах, что и размер типа.

```
int arri[3] = {1,2,3};
int *ptrs = &arri[0];
int *ptre = &arri[2];
printf("%p - %p = %d\n",ptrs,ptre,ptre-ptrs);
```

```
0000008F975FF944 - 0000008F975FF94C = 2
```

Вычитание является гарантированно допустимой операцией при условии, что оба *указателя* ссылаются на значения внутри одного и того же массива (или, возможно, на позицию за последним элементом массива). Применение этой операции к указателям в двух разных массивах может дать какое-то значение или привести к ошибке во время выполнения.

Операции с указателями (8)

Сравнение. Для сравнения значений двух указателей можно использовать операции отношений при условии, что указатели имеют один и тот же тип.

```
int arri[3] = {1,2,3};
printf("%d\n", (&arri[0] == arri));
printf("%d\n", (&arri[3] < &arri[1]));
printf("%d\n", (&arri[2] > &arri[2]));
printf("%d\n", (&arri[3] > &arri[0]));
```

```
1
0
0
1
```

Плохие операции с указателями (9)

Разыменовывание неинициализированного указателя.

Не стоит допускать следующей операции.

```
int * pt;
*pt = 5;
```

Показанный код сохраняет значения 5 в *pt. pt не инициализирован, а значит имеет случайное значение, возникает состояние неопределенности, так как нет информации о том, куда будет сохранено значение 5.

Защита содержимого массива

Как видно из примеров приведенных ранее, работа с *массивами* и *указателями* содержит множества тонкостей. Возникают ситуации в которых нет никакой потребности изменять значения массива, а следует лишь “прочитать” значения.

```
int sum(const int ar[] , int n); /* прототип */
…
int sum(const int ar[], int n) /* определение */
{
int i; int total = 0;
for ( i = 0; i < n; i + + )
total += ar [ i ] ;
return total;}
```

Ключевое слово const (дополнение) (1)

Вам знакомо применение **const** на уровне создание констант. В отличии от **#define, const** предоставляет больше возможностей.

```
const int days[MONTHS]=
{31,28,31,30,31,30,31,31,30,31,30,31};
days[9]=44; //Ошибка компиляции
double rates[5] = {1.,2.,3.,4.,5.};
const double *ptrd = rates;
*ptrd = 6.; //Запретная операция
ptrd[3]=7.; //Запретная операция
rates[4]=8.; //Не вызывает проблем
*ptrd++; //Переводим указатель на rates[1]
```

Ключевое слово const (дополнение) (2)

Модификатор данных **const**, может быть использован для создания *указателя*, который нельзя заставить *указывать* на что-нибудь другое, кроме как на инициализированное значение.

```
int num[3] = {2,4,8};
int * const ptri = num; //указатель на num[0]
ptri = &num[2]; //запрещено
*ptri = 100; //разрешено
```

Ключевое слово const (дополнение) (3)

Модификатор данных const, может быть использован для создания указателя, который нельзя заставить указывать на что-нибудь другое, а также изменять значение на которое он указывает.

```
int num[3] = {2,4,8};
const int * const ptri = num; //указатель на num[0]
ptri = &num[2]; //запрещено
*ptri = 100; //запрещено
```

Многомерные массивы и указатели (1)

Многомерность усложняет восприятие указателей.

```
int tab[4][2] = {{3,6},{2,4},{8,1},{9,0}};
```

Имя	-	tab	-	-	-	-	-	-	-	-
Одномерное		tab[0]		tab[1]		tab[2]		tab[3]		
Двумерное		tab[0][0]	tab[0][1]	tab[1][0]	tab[1][1]	tab[2][0]	tab[2][1]	tab[3][0]	tab[3][1]	
Адрес	0x04	0x08	0x0C	0x10	0x14	0x18	0x1C	0x20	0x24	0x28
Значение	2453	3	6	2	4	8	1	9	0	8
Индекс	-	0	1	2	3	0	1	2	3	0

```
printf
("%p",tab+1)     ("%d",tab[0][0]) ("%d",tab[2][1])
("%p",tab[0]+1)  ("%d",*tab[0]) ("%d",*(*(tab+2)+1)
                 ("%d",**tab)
```

Многомерные массивы и указатели (2)

Многомерность усложняет восприятие указателей.

```
int tab[4][2] = {{3,6},{2,4},{8,1},{9,0}};
```

Имя	-	tab	-	-	-	-	-	-	-	-
Одномерное		tab[0]		tab[1]		tab[2]		tab[3]		
Двумерное		tab[0][0]	tab[0][1]	tab[1][0]	tab[1][1]	tab[2][0]	tab[2][1]	tab[3][0]	tab[3][1]	
Адрес	0x04	0x08	0x0C	0x10	0x14	0x18	0x1C	0x20	0x24	0x28
Значение	2453	3	6	2	4	8	1	9	0	8
Индекс	-	0	1	2	3	0	1	2	3	0

```
printf
("%p",tab+1)        ("%d",tab[0][0])  ("%d",tab[2][1])
("%p",tab[0]+1)     ("%d",*tab[0])    ("%d",*(*(tab+2)+1)
                    ("%d",**tab)
```

Многомерные массивы и указатели (3)

```
int tab[4][2] = {{3,6},{2,4},{8,1},{9,0}};
*tab = int [2]
**tab = int
```

Доступа к различным значениям многомерного массива можно использовать обе записи:

```
tab[m][n] == *(*(tab+m) + n)
```

Если вы создаете *указатель* на двумерный массив следует учитывать приоритеты операций: **оператор []** более высокий приоритет чем **оператор ***, таким образом приоритет операции стоит указывать явно используя **оператор ().**

```
int (*tab)[2];
```

Многомерные массивы и функции (1)

Для обработки двумерных массивов используются комбинации циклов для одномерных массивов.

```
#define ROWS 3
#define COLS 4
int main(void){
int junk[ROWS][COLS] = {
{2,4,6,8},{3,5,7,9},{12,10,8,6}};
sum_rows (junk, ROWS);
sum_cols (junk, ROWS);
sum2d(junk, ROWS);
return 0;}
```

Многомерные массивы и функции (2)

```
void sum_rows(int ar[][COLS], int rows);
…
void sum_rows(int ar[][COLS], int rows){
int r, c, tot;
for (r =0; r < rows; r++){
    tot = 0;
    for (c = 0; c < COLS; c++) tot += ar[r][c];
    printf("строка %d: сумма = %d\n", r, tot);
    }
}
```

Многомерные массивы и функции (3)

```
void sum_cols(int [][COLS], int );
...
void sum_cols(int ar[][COLS], int rows){
int r, c , tot;
    for (c =0; c < COLS; c++)
    {
      tot = 0;
      for (r =0; r < rows; r++)
      tot += ar[r][c];
      printf ("столбец %d: сумма = %d\n", c, tot);
    }
}
```

Многомерные массивы и функции (4)

```
void sum2d(int (*ar)[COLS], int rows);
...
void sum2d(int ar[][COLS], int rows){
int r;
int c;
int tot = 0;
for (r = 0; r < rows; r++)
    for (c = 0; c < COLS; c++)
        tot += ar[r][c];
printf ("сумма всех элементов = %d\n", tot);
}
```

Массивы переменной длинны (1)

Мотивация для ввода в стандарт C99 понятия массивов переменной длинны, обоснована тем, что язык Си является приемником FORTRAN, и это необходимо для совместимости с его библиотеками.

```
int quarters = 4;
int regions = 5;
double sales[regions][quarters];
```

Понятие переменный в массиве переменной длины вовсе не означает возможность изменения длины массива после его создания. Будучи созданным, массив переменной длины сохраняет тот же самый размер. В действительности понятие переменный означает, что при указании размерностей при первоначальном создании массива можно использовать переменные.

Массивы переменной длинны (1)

```
int quarters = 4;
int regions = 5;
double sales[regions][quarters];
int sum2d(int rows, int cols, int ar[rows][cols]);
```

Обратите внимание на то, что два первых параметра (rows и cols) применяются в качестве размерностей для объявления параметра типа массива ar. Поскольку в объявлении ar используются rows и cols, в списке параметров они должны быть объявлены до появления ar. Поэтому следующий прототип является ошибочным:

```
int sum2d(int ar[rows][cols], int rows, int cols);
```

Массивы переменной длинны (2)

```
int sum2d(int, int, int ar[*][*]);
…
int sum2d(int rows, int cols, int ar[rows][cols])
{
int r = 0, c = 0, tot = 0;
for (r = 0; r < rows; r++)
    for (c = 0 ; c < cols; c++)
        tot += ar[r][c];
return tot;
}
```

Составные литералы

До выхода стандарта C99 положение дел с функцией, принимающей аргумент типа массива, было другим; можно было передавать массив, но отсутствовал эквивалент для константы типа массива.

```
[1](int [2]) {50, 100}
[2](int []) {2,4,8,16}
[3] int * ptl;
    ptl = (int [2]) {10, 20);
[4] int sum(const int ar[], int n) ;
    int total;
    total = sum ((int []) {4,4,4,5,5,5), 6);
[5] int (*pt2) [4]);
pt2 = (int [2][2]) {{3,-9), {6,-8)};
```

Задачи на практику

Читать:

Главу 10. Массивы и указатели. (стр 384-413)

Вопросы для самоконтроля (стр 414)

Изучить:

Совместимость указателей

Практическая работа:

ПР 9. (Зачёт) Многомерные массивы

Спасибо за внимание

Ревун Артем Леонидович
ст. преп. Кафедры вычислительных систем