

Практикум на ЭВМ

Динамические многомерные массивы. Динамические структуры.

Баев А.Ж.

Казахстанский филиал МГУ

11 октября 2020

План

- 1 Статические матрицы
- 2 Динамические матрицы
- 3 Аргументы
- 4 Стрелка
- 5 Список
- 6 gdb
- 7 sanitizers

Статические матрицы

```
1 int a[2];  
2 int b[2][3][4];
```

Переменная `a` является указателем типа `int *`.

Переменная `b` является указателем типа `int (*)[3][4]`.

При передачи в функцию, статические массивы не передаются по значению — передаются соответствующие указатели. Следующие прототипы одинаковы:

```
1 void f(int array[2][3][4]);  
2 void f(int (*array)[3][4]);
```

Почему можно опустить первую размерность?

Статические матрицы

Обратите внимание на круглые скобки.

```
1  int (*array)[3][4]
2  (int *) array[3][4]
3  int *array[3][4]
```

Статические матрицы — плотно упакованные

```
1 void *memset(void *s, int c, size_t n);  
2 void *memcpy(void *dest, void *src, size_t n);
```

Инициализация всей матрицы.

```
1 #include <string.h>  
2  
3     const int n = 10, m = 20;  
4  
5     int A[n][m];  
6     memset(A, 0, n * m * sizeof(int));  
7  
8     int B[n][m];  
9     memcpy(B, A, n * m * sizeof(int));  
10  
11     int C[n * m];  
12     memcpy(C, A, n * m * sizeof(int));
```

Динамические матрицы

- 1) уложить всю матрицу построчно в виде одного динамического массива (плотная упаковка);
 - 2) создать дополнительный массив указателей на динамические массивы (неплотная упаковка).
- Смотрим второе. Почему?

Динамические матрицы. Матрица 2x3

Каждая строка матрицы — динамический массив. Где хранить указатели на эти строки? Еще в одном динамическом массиве!

```
1  int **a = malloc(2 * sizeof(int *));
```

Статическая

Переменная
Значение

a short **
0x2000

Динамическая

Адрес
Обращение
Значение

0x2000	0x2004
a[0] short *	a[1] short *
???	???

Динамические матрицы. Матрица 2x3

```
1   a[0] = malloc(3 * sizeof(int));  
2   a[1] = malloc(3 * sizeof(int));
```


Динамические матрицы. Матрица 2x3

Статическая

Переменная

Значение

a short **
0x2000

Динамическая

Адрес

Обращение

Значение

0x2000	0x2004
a[0] short *	a[1] short *
0x5000	0x3000

Адрес

Обращение

Значение

0x5000	0x5002	0x5004
a[0][0]	a[0][1]	a[0][2]
1	2	3

Адрес

Обращение

Значение

0x3000	0x3002	0x3004
a[1][0]	a[1][1]	a[1][2]
4	5	6

Динамические матрицы. Матрица 2x3

Очистка

```
1    free(a[0]);  
2    free(a[1]);  
3    free(a);
```

Динамические матрицы. Любимая плюшка линала

Перестановка строк!

```
1      short *tmp = a[0];  
2      a[0] = a[1];  
3      a[1] = tmp;
```

Динамические матрицы. Матрица 2x3

Статическая

Переменная

Значение

a short **
0x2000

Динамическая

Адрес

Обращение

Значение

0x2000	0x2004
a[0] short *	a[1] short *
0x3000	0x5000

Адрес

Обращение

Значение

0x5000	0x5002	0x5004
a[1][0]	a[1][1]	a[1][2]
1	2	3

Обращение

Адрес

Значение

0x3000	0x3002	0x3004
a[1][0]	a[1][1]	a[1][2]
4	5	6

Отличия динамической и статической матрицы.

1. матрица размера $n \times m$, где каждый элемент имеет размер s , а указатель — размер p . В случае со статической памятью: $n \cdot m \cdot s$. В случае с динамической памятью: $(1 + n) \cdot p$ для указателей и $n \cdot m \cdot s$ для самой матрицы.
2. динамические матрицы хранятся кусками по строкам, а статические — цельном блоком. Перестановка строк $O(1)$.
3. размеры разных строк могут быть разными.

Подумать.

1. Какие присваивания допустимы между указанными переменными (среди всех возможных пар)?

```
1      int a[2][3];  
2      int (*b)[3];  
3      int *c[3];  
4      int **d;
```

2. Динамическая матрица из 3×4 . Строки и столбцы — динамические.

3. Динамическая матрица из 3×4 . Строки — статические, столбцы — динамические.

4. Динамическая матрица из 3×4 . Строки — динамические, столбцы — статические.

Аргументы командной строки.

```
1 $ gcc prog.c -o prog -Wall
2 $ ./prog sum 1 2.0 3e4
```

Здесь 4 аргумента:

```
1 int main(int argc, char** argv);
```

При запуске вашей программы, параметр `argc` будет содержать количество параметров командной строки, включая название программы (5).

Аргумент `argv` — это массив строк, которые сформированы из аргументов, включая название программы:

```
1 argv[0] == "./prog"
2 argv[1] == "sum"
3 argv[2] == "1"
4 argv[3] == "2.0"
5 argv[4] == "3e4"
```

Перевод строковых данных.

```
1 #include <stdlib.h>
2
3 double atof(const char *nptr);
4 int atoi(const char *nptr);
5 long atol(const char *nptr);
6 long long atoll(const char *nptr);
```

```
1 #include <stdio.h>
2
3 int sprintf(char *str, const char *format, ...);
```


Подумать.

1. Написать программу, которая печатает свой исходный код, при условии, что он лежит рядом и имеет такое же название (prog и prog.c).
2. Написать программу, которая печатает свой бинарный код.

Указатель на структуру.

Оператор звезда и точка или оператор стрелка.

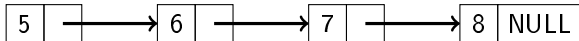
```
1 struct Point {  
2     int x, y;  
3 };  
4 ...  
5 struct Point point = {2, 3};  
6 struct Point *ptr = &point;  
7 (*ptr).x = 5;  
8 ptr->y = 6;
```

Указатель на структуру.

Оператор «звезда и точка» или оператор «стрелка».

```
1 struct Point {  
2     int x, y;  
3 };  
4 ...  
5     struct Point point = malloc(sizeof(struct Point))  
6     (*ptr).x = 5;  
7     ptr->y = 6;
```

Динамическая структура — список.



```
1  typedef struct node *link;
2
3  struct node {
4      int elem;
5      link next;
6  };
7
8  int main() {
9      link list = NULL;
10     list = push_front(list, 6);
11     list = push_front(list, 5);
12     list = push_back(list, 7);
13     list = push_back(list, 8);
14     return 0;
15 }
```

Список. Добавить в начало.

ptr

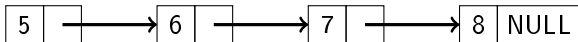


ptr



ptr

list



```
1 link push_front(link list, int elem) {  
2     link ptr = malloc(sizeof(struct node));  
3     ptr->elem = elem;  
4     ptr->next = list;  
5     return ptr;  
6 }
```

Список. Типичная ошибка.

ptr

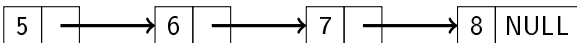


ptr



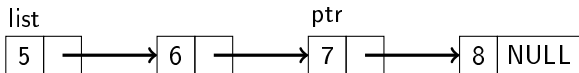
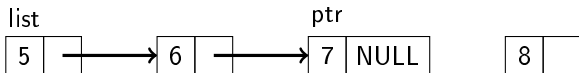
ptr

list



```
1 void push_front(link list, int elem) {  
2     link ptr = malloc(sizeof(struct node));  
3     ptr->elem = elem;  
4     ptr->next = list;  
5     list = ptr;  
6 }
```

Список. Добавить в конец.

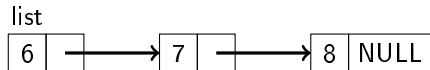
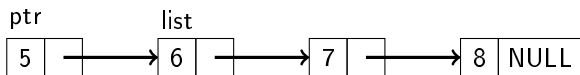


```

1 link push_back(link list, int elem) {
2     if (list == NULL) {
3         return push_front(list, elem);
4     }
5     link ptr = list;
6     while (ptr -> next != NULL) {
7         ptr = ptr -> next;
8     }
9     ptr->next = push_front(NULL, elem);
10    return list;
11 }

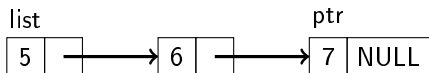
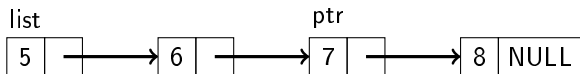
```

Список. Убрать из начала.



```
1 link pop_front(link ptr) {
2     link list = ptr->next;
3     free(ptr);
4     return list;
5 }
```


Список. Убрать из конца.



```
1 link pop_back(link list) {  
2     if (list -> next == NULL) {  
3         return pop_front(list);  
4     }  
5     link ptr = list;  
6     while (ptr -> next -> next != NULL) {  
7         ptr = ptr -> next;  
8     }  
9     ptr->next = pop_front(ptr->next);  
10    return ptr;  
11 }
```

Список. Вывод. Очистка.

```
1 void print(link list) {  
2     putchar('[');  
3     while (list != NULL) {  
4         printf("%d_", list->elem);  
5         list = list -> next;  
6     }  
7     puts("]");  
8 }
```

```
1 link clear(link list) {  
2     link ptr;  
3     while (list != NULL) {  
4         ptr = list -> next;  
5         free(list);  
6         list = ptr;  
7     }  
8     return NULL;  
9 }
```

Список. Генерируем список.

```
1 link range(int from, int to) {  
2     link list = NULL;  
3     int elem = to - 1;  
4     while (elem >= from) {  
5         list = push_front(list, elem);  
6         elem--;  
7     }  
8     return list;  
9 }
```

Отладка в gdb

Компилируем

```
1 gcc prog.c -o prog -Wall -Werror -lm -g
```

Запускаем отладчик

```
1 gdb prog
```

Посмотрим код

```
1      list 1
```

Ставим точку останова (до которой программа будет выполняться в обычном режиме). Лучше ставить сразу после ввода.

```
1      break 6
```

Запускаем

```
1      run
```

Добавляем переменную наблюдения (можно несколько переменных)

```
1      display n  
2      display ans
```

Делаем построчное выполнение (первый раз надо набрать команду целиком, потом просто Enter).

```
1      next
```

Выход

```
1      quit
```

Отладка в gdb

```
1 79      int main() {  
2 80          link list = NULL;  
3 81          list = push_front(list, 5);  
4 82          print(list);  
5 83          list = clear(list);  
6 84          return 0;  
7 85      }
```

```
1 (gdb) break 80  
2 Breakpoint 1 at 0x93e: file prog.c, line 80.
```

```
1 (gdb) run  
2 Starting program: prog  
3 Breakpoint 1, main () at prog.c:80  
4 80          link list = NULL;
```

```
1 (gdb) display list  
2 1: list = (link) 0x0
```

Отладка в gdb

```
1 79      int main() {  
2 80          link list = NULL;  
3 81          list = push_front(list, 5);  
4 82          print(list);  
5 83          list = clear(list);  
6 84          return 0;  
7 85      }
```

```
1 (gdb) next  
2 81          list = push_front(list, 5);  
3 1: list = (link) 0x0
```

```
1 (gdb)  
2 82          print(list);  
3 1: list = (link) 0x555555756260
```

Отладка в gdb

```
1 79      int main() {  
2 80          link list = NULL;  
3 81          list = push_front(list, 5);  
4 82          print(list);  
5 83          list = clear(list);  
6 84          return 0;  
7 85      }
```

```
1 (gdb)  
2 [5 ]  
3 83      list = clear(list);  
4 1: list = (link) 0x555555756260
```

```
1 (gdb)  
2 84      return 0;  
3 1: list = (link) 0x0
```


Отладка в gdb. Без ошибок.

```
1 79      int main() {
2 80          link list = NULL;
3 81          list = push_front(list, 5);
4 82          print(list);
5 83          list = clear(list);
6 84          return 0;
7 85      }
```

```
1 (gdb) break 80
2 Breakpoint 1 at 0x93e: file prog.c, line 80.
```

```
1 (gdb) run
2 Starting program: prog
3 Breakpoint 1, main () at prog.c:80
4 80          link list = NULL;
```

```
1 (gdb) display list
2 1: list = (link) 0x0
```

Отладка в gdb. Без ошибок.

```
1 79      int main() {  
2 80          link list = NULL;  
3 81          list = push_front(list, 5);  
4 82          print(list);  
5 83          list = clear(list);  
6 84          return 0;  
7 85      }
```

```
1 (gdb) next  
2 81          list = push_front(list, 5);  
3 1: list = (link) 0x0
```

```
1 (gdb)  
2 82          print(list);  
3 1: list = (link) 0x555555756260
```

Отладка в gdb. Без ошибок.

```
1 79      int main() {  
2 80          link list = NULL;  
3 81          list = push_front(list, 5);  
4 82          print(list);  
5 83          list = clear(list);  
6 84          return 0;  
7 85      }
```

```
1 (gdb)  
2 [5 ]  
3 83      list = clear(list);  
4 1: list = (link) 0x555555756260
```

```
1 (gdb)  
2 84      return 0;  
3 1: list = (link) 0x0
```

Отладка в gdb. Ошибки.

```
1 79      int main() {  
2 80          link list = NULL;  
3 81          list = pop_front(list);  
4 82          print(list);  
5 83          list = clear(list);  
6 84          return 0;  
7 85      }
```

```
1 (gdb) break 80  
2 Breakpoint 1 at 0x93e: file prog.c, line 80.
```

```
1 (gdb) run  
2 Starting program: prog  
3 Breakpoint 1, main () at prog.c:80  
4 80          link list = NULL;
```

```
1 (gdb) display list  
2 1: list = (link) 0x0
```

Отладка в gdb. Ошибки.

```
1 79      int main() {
2 80          link list = NULL;
3 81          list = pop_front(list);
4 82          print(list);
5 83          list = clear(list);
6 84          return 0;
7 85      }
```

```
1 (gdb) next
2 81          list = pop_front(list);
3 1: list = (link) 0x0
```

```
1 (gdb)
2 Program received signal SIGSEGV,
3 Segmentation fault.
4 0x0000555555554762 in pop_front (ptr=0x0)
5 at prog.c:19
6 19          link list = ptr->next;
```

Санитайзер

```
1 gcc prog.c -o prog -fsanitize=address,leak
```

```
1 ==11053==ERROR: AddressSanitizer: SEGV on unknown
2   address 0x00000000000008 (pc 0x55db755ccbfb
3   bp 0x7fff74e0dfc0 sp 0x7fff74e0dfa0 T0)
4 ==11053==The signal is caused by
5   a READ memory access.
6 ==11053==Hint: address points to the zero page.
7   #0 0x55db755ccbfa in pop_front ./prog.c:19
8   #1 0x55db755ccf91 in main ./prog.c:81
9   #2 0x7f5fc8a69b96 in __libc_start_main
10      (/lib/x86_64-linux-gnu/libc.so.6+0x21b96)
11   #3 0x55db755cca59 in _start
12      (./prog+0xa59)
13 AddressSanitizer can not provide additional info.
14 SUMMARY: AddressSanitizer:
15   SEGV ./prog.c:19 in pop_front
16 ==11053==ABORTING
```