Динамическая память

Основы языка C, лекция 9

Повторяем

- Дано число N (0 < N ≤1000)
  далее N целых чисел. Напечатать их дважды.

- ```
  #define N 1000
  int main ( ) {
      int a[N], n, i;
      scanf("%d", &n);
      for (i = 0; i < n; i++)
          scanf("%d", &a[i]);       // a+i

      print_arr(a, n);    print_arr(a, n);
      return 0;
  }
  ```

```
5
7 -1 3 12 8
```
```

C99

- Дано число N (0 < N ≤1000)
  далее N целых чисел. Напечатать их дважды.

- ```
  int main ( ) {
      int n;
      scanf("%d", &n);
      int a[n];
      for (int i = 0; i < n; i++)
          scanf("%d", &a[i]);       // a+i

      print_arr(a, n);    print_arr(a, n);
      return 0;
  }
  ```

```
5
7 -1 3 12 8
```
```

N не дано

- ~~Дано число N~~ (0 < N ≤1000)
  далее N целых чисел. Напечатать их дважды.

- ```
  #define N 1000
  int main ( ) {
      int a[N], n, i;
      scanf("%d", &n);
      for (i = 0; i < ?; i++)
          scanf("%d", &a[i]);       // a+i

      print_arr(a, n);    print_arr(a, n);
      return 0;
  }
  ```

```
7 -1 3 12 8
```
```

Что возвращает scanf

- scanf возвращает количество правильно разобранных format placeholder
- int x, y, res; res = scanf("%d%d", &x, &y);
- while (1) {
  res = scanf (
  "%d%d", &x, &y);
  if (res != 2)
  break;
  ....
  }

```
12 34
-6 15
88 -21
```

input	res	x	y
12 34	2	12	34
13abc	1	13	?
abc25	0	?	?
hello	0	?	?
конец файла	EOF	?	?

N не дано, но ограничено

- ~~Дано число N~~ (0 < N ≤1000)
  далее N целых чисел. Напечатать их дважды.

- ```
  #define N 1000
  int main ( ) {
      int a[N], n = 0;
      while (1 == scanf("%d", a+n ) )
          n++;

      print_arr(a, n);    print_arr(a, n);
      return 0;
  }
  ```

```
7 -1 3 12 8
```
```

не знаем чем ограничено N

- Дано число N ~~(0 < N ≤1000)~~
  далее N целых чисел. Напечатать их дважды.

- ```
  int main ( ) {
      int n;
      scanf("%d", &n);
      int a[n];
      for (int i = 0; i < n; i++)
          scanf("%d", &a[i]);       // a+i

      print_arr(a, n);    print_arr(a, n);
      return 0;
  }
  ```

```
5
7 -1 3 12 8
```
```

Без C99

- Дано число N ~~(0 < N ≤1000)~~
  далее N целых чисел. Напечатать их дважды.

```
5
7 -1 3 12 8
```

```
int main ( ) {
    int n;
    scanf("%d", &n);
    // int a[n];
    int * a = malloc( n * sizeof(int));  // взяли
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);       // a+i
    print_arr(a, n);    print_arr(a, n);
    free (a);                                  // отдали
    return 0;
}
```

Функции работы с памятью

- #include <stdlib.h>
  void * **malloc** (size_t **size**);
  void * **calloc** (size_t **nmemb**, size_t **size**);
  void * **realloc** (void * **ptr**, size_t **size**);
  void **free** (void * **ptr**);
- malloc — выделить память размером size и вернуть на нее указатель
- calloc — выделить память размером nmemb*size, заполнить ее 0, вернуть указатель на память
  a = **calloc** (n, sizeof(int));
- free — вернуть выделенную память ptr

free (ptr)

- указатель на **динамически** выделенную память

  int * a = malloc(10); .... ; free(a); // ok

  char b[10]; .... free(b); // error

- указатель на **начало** выделенной памяти

  int * a = malloc(10); int * b = malloc(10);

  - free(a); // ok
  - free(b +2); // error

- free (**NULL**); - ничего не делает
- free(a); free(a); - undefined behaviour

Ничего о N неизвестно

- Дано N целых чисел. Напечатать их дважды.
- int main ( ) {

```
    int n;                                                    7 -1 3 12 8
    int * a = malloc( 1 * sizeof(int));  // взяли для 1 int
    while (1) {
        if (scanf("%d", a+n) != 1)      // &a[n]
            break;
        n ++;
        a = realloc(a, (n+1)*sizeof(int));   // взяли новое
    }
    print_arr(a, n);    print_arr(a, n);
    free (a);                                  // отдали
```

newptr = realloc (ptr, size)

- захватить память размером size
- откопировать в нее память из ptr
  (если размеры разные?)
- освободить старую память, на которую указывал ptr
- вернуть указатель на новую память

realloc(NULL, size) как malloc

- Дано N целых чисел. Напечатать их дважды.
- int main ( ) {

```
    int n = 0;
    int * a = NULL;
    while (1) {
        a = realloc(a, (n+1)*sizeof(int));   // взяли новое
        if (scanf("%d", a+n) != 1)      // &a[n]
            break;
        n ++;
    }
    print_arr(a, n);    print_arr(a, n);
    free (a);                                   // отдали
```

7 -1 3 12 8

nsize — выделено памяти

7 -1 3 12 8

```
int main ( ) {
    int n = 0;    // чисел прочитали
    int nsize = 10;  // выделено памяти под nsize чисел
    int * a = malloc(a, (nsize)*sizeof(int));
    while (1) {
        if (scanf("%d", a+n) != 1)      // &a[n]
            break;
        n ++;
        if (n >= nsize) {
            nsize += 10;            // увеличим
            a = realloc(a, nsize*sizeof(int));
        }
    }
```

Единообразно

7 -1 3 12 8

```
int main ( ) {
    int n = 0;    // чисел прочитали
    int nsize = 0;  // выделено памяти под nsize чисел
    int * a = NULL;
    while (1) {
        if (n >= nsize) {
            nsize += 10;            // увеличим
            a = realloc(a, nsize*sizeof(int));
        }
        if (scanf("%d", a+n) != 1)      // &a[n]
            break;
        n ++;
    }
```

функции работы с памятью

- #include <string.h>
  void * **memset** (void ***s**, int **c**, size_t **n**);
  от указателя **s** заполнить **n** байт значением **c**.
- заполнить массив a нулями:
  int a[10];
  memset(a, 0, sizeof(a));
- можно ли написать memset, который делает то же самое (записывает 10 единиц)?
  for(int i = 0; i < 10; i++)
  a[i] = 1;

функции работы с памятью

- #include <string.h>
  char * **strcpy** (char ***dst**, const char * **src**);
  void * **memcpy** (void ***dst**, const void * **src**, size_t **n**);
  void * **memmove** (void ***dst**, const void * **src**, size_t **n**);
- strcpy — из указателя src копируем в указатель dst **до \0**, возвращаем dst;
- memcpy — из указателя src копируем в указатель dst **n байт**, возвращаем dst;
- memmove — из указателя src копируем в указатель dst **n байт, можно перекрывающиеся участки памяти**, возвращаем dst; - копирование через буфер

void * в разных языках

- C: автоматическое преобразование указателей к/из void *

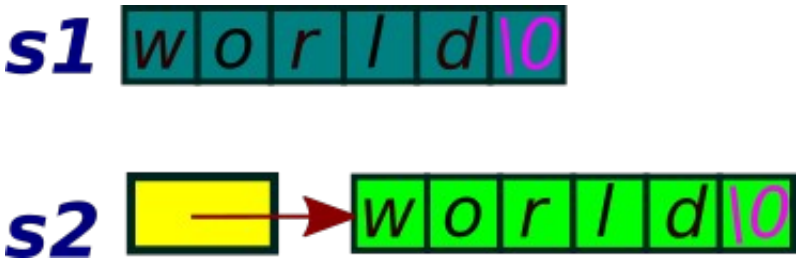
```
char * a = malloc(6);        // void * → char *
a = realloc(a, 12);          // char * →void *
```

- C++ типы НЕ преобразуются автоматически

```
char * a = (char *) malloc(6);
a = (char *) realloc( (void*)a, 12);
```

Выделение памяти под строку

- char s1[ ] = "world";
  strlen(s1); // 5
  sizeof(s1); // 6



- char * s2 = "world";
  strlen(s2); // 5
  sizeof(s2); // 8 на 64-бит
- char * str = malloc(strlen(s2) + 1);
  strcpy(str, s2);
- char * str = **strdup** (s2);

Прочитать 1 слово

- char * name;
  scanf ("**%ms**", &name);
  ...
  free (name);

Прочитать 1 строку

- #include <stdio.h>
  ssize_t **getline** (char ****lineptr**, size_t ***n**, FILE ***stream**);
- ssize_t **getdelim**(char ****lineptr**, size_t ***n**, int **delim**, FILE ***stream**);
- getline() считывает целую строку, сохраняя адрес буфера, содержащего текст, в *lineptr. Буфер завершается null и содержит символ новой строки, если был найден разделитель для новой строки.
- Если *lineptr равно NULL, то процедура getline() будет создавать буфер для содержимого строки, который затем должен быть высвобожден программой пользователя.

Прочитать 1 строку — 2

- ssize_t **getline** (char ****lineptr**, size_t ***n**, FILE ***stream**);
- ssize_t **getdelim**(char ****lineptr**, size_t ***n**, int **delim**, FILE ***stream**);
- Как альтернатива, перед вызовом getline(), *lineptr может содержать указатель на буфер, размещенный через malloc() с размером *n байтов. Если буфер недостаточно велик для размещения всей считанной строки, то getline() изменяет размер буфера с помощью realloc(), обновляя *lineptr и *n при необходимости. В любом случае при успешном вызове *lineptr и *n будут обновлены для отражения адреса буфера и его размера соответственно.

Прочитать 1 строку — 3

- getdelim() работает аналогично getline(), за исключением того, что разделитель строки, отличающийся от символа новой строки будет определен, как аргумент delimiter. Как и с getline(), символ-разделитель не добавляется, если на вводе не появилось знака разделения и уже достигнут конец файла.
- При нормальном завершении работы getline() и getdelim() возвращают номер считанных символов, включая символ разделителя, но не включая завершающий символ null. Это значение может использоваться для обработки встроенных символов null при чтении строки.

Прочитать 1 строку — 4

- Обе функции возвращают -1 при ошибках чтения строки (включая условие достижения конца файла).
- char * line = NULL;
  size_t len = 0;
  FILE * fp = fopen("/etc/motd", "r");
  if (fp == NULL)
      exit(EXIT_FAILURE);
  ssize_t read;
  while ((read = getline(&line, &len, fp)) != -1) {
      printf("Retrieved line of length %zu :\n", read);
      printf("%s", line);
  }
  if (line)

Прочитать по 1 строке пример

```
char * line = NULL;
size_t len = 0;
FILE * fp = fopen("/etc/motd", "r");
if (fp == NULL)
    exit(EXIT_FAILURE);
ssize_t read;
while ((read = getline (&line, &len, fp)) != -1) {
    printf("Retrieved line of length %zu :\n", read);
    printf("%s", line);
}
if (line)
    free (line);
return EXIT_SUCCESS;
```

valgrind

матрица пример