Содержание

stdlib.h

Управление динамической памятью

malloc

прототип: `void *malloc(unsigned s);`

Функция malloc() выделяет память длиной для определенного количества байт и возвращает указатель на начало выделенной памяти. Через полученный указатель мы можем помещать данные в выделенную память. Рассмотрим простой пример:

```
#include <stdio.h>
#include <stdlib.h>   // для подключения функции malloc

int main(void) {
  int *ptr = malloc(sizeof(int)); // выделяем память для одного int
  if(ptr != NULL) {
    *ptr = 24;      // помещаем значение в выделенную память
    printf("%d \n", *ptr);
  }
  free(ptr);
  ptr = NULL;
}
```

Немотря на освобождение памяти с помощью функции free() указатель сохраняет свой адрес, и теоретически мы можем обращаться к памяти по данному указателю. Однако полученные значения уже будут неопределенными и недетеминированными. Поэтому некоторые советуют после освобождения памяти также устанавливать для указателя значение NULL

выделение памяти под массив

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  int n = 4;
  int *ptr = malloc(n * sizeof(int)); // выделяем память для 4-х чисел int
  if(ptr) {
    // помещаем значения в выделенную память
    ptr[0] = 1;
    ptr[1] = 2;
    ptr[2] = 3;
    ptr[3] = 5;
```

```
    // получаем значения
    for(int i = 0; i < n; i++) {
      printf("%d", ptr[i]);
    }
  }

  free(ptr);
}
```

выделение памяти под структуру

```
#include <stdio.h>
#include <stdlib.h>

struct person{
  char* name;
  int age;
};

int main(void) {
  // выделяем память для одной структуры  person
  struct person *ptr = malloc(sizeof(struct person));
  if(ptr) {
    // помещаем значения в выделенную память
    ptr->name = "Tom";
    ptr->age = 38;

    // получаем значения
    printf("%s : %d", ptr->name, ptr->age);       // Tom : 38
  }

  free(ptr);
  return 0;
}
```

calloc

прототип: `void *calloc(unsigned n, unsigned m);`

Она выделяет память для n элементов по m байт каждый и возвращает указатель на начало выделенной памяти. В случае неудачного выполнения возвращает NULL

В отличие от функции malloc() она инициализирует все выделенные байты памяти нулями.

realloc

прототип: `void *realloc(void *bl, unsigned ns);`

Первый параметр представляет указатель на ранее выделенный блок памяти. А второй параметр представляет новый размер блока памяти в байтах.

Если указатель bl имеет значение NULL, то действие функции аналогично действию malloc

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  // выделяем память для 1-го объекта int
  int size = sizeof(int);
  int *ptr = malloc(size);
  if(ptr) {
    // отображаем адрес и размер памяти
    printf("Addresss: %p \t Size: %d\n", (void*)ptr, size);
  }
  // расширяем память до размера 4-х объектов int
  size = 4 * sizeof(int);
  int *ptr_new = realloc(ptr, size);
  // если выделение памяти прошло успещно
  if(ptr_new) {
    printf("Reallocation\n");
    // заново отображаем адрес и размер памяти
    printf("Addresss: %p \t Size: %d\n", (void*)ptr_new, size);
    free(ptr_new);   // освобождаем новый указатель
  } else {
    free(ptr);   // освобождаем старый указатель
  }
}
```

free

Имеет прототип: `void *free(void *bl);`

Освобождает ранее выделенный блок памяти, на начало которого указывает указатель bl.

Расмотрим, когда нужно освобождать память:

- Указатель определен в блоке кода. В этом случае указатель будет доступен только в пределах данного блока кода. Соответственно память необходимо освобождать при выходе из этого блока.
- Указатель определен как статический объект. В этом случае динамическая память выделяется один раз и доступна через указатель при каждом повторном входе блок. В этом случае память нужно освобождать только после завершения ее использования.
- Указатель является глобальным объектом по отношению к блоку. В этом случае динамическая память доступна во всех блоках, где доступен указатель, а память нужно освобождать только после завершения ее использования.