Динамическое распределение памяти в C с использованием malloc, calloc, free и realloc

Поскольку C — структурированный язык, в нем действуют некоторые фиксированные правила программирования. Один из них включает в себя изменение размера массива. Массив — это совокупность элементов, хранящихся в смежных ячейках памяти (см. Рис. 1).

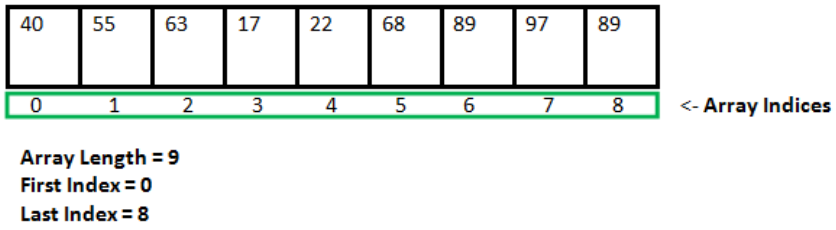

Рис. 1. Массив

Как видно, длина (размер) массива выше равна 9. Но что, если возникнет требование изменить эту длину (размер)? Например,

- Если возникла ситуация, когда в этот массив необходимо ввести только 5 элементов. В этом случае оставшиеся 4 индекса просто тратят память в этом массиве. Поэтому существует требование уменьшить длину (размер) массива с 9 до 5.
- Возьмем другую ситуацию. Здесь имеется массив из 9 элементов со всеми 9 заполненными индексами. Но в этот массив необходимо ввести еще 3 элемента. В этом случае потребуется еще 3 индекса. Значит длину (размер) массива нужно изменить с 9 на 12.

Эта процедура называется динамическим распределением памяти в C. Таким образом, динамическое распределение памяти.

C можно определить как процедуру, в которой размер структуры данных (например, массива) изменяется во время выполнения. C предоставляет некоторые функции для решения этих задач. Существует 4 библиотечные функции, предоставляемые C, определенные в заголовочном файле <stdlib.h> для облегчения динамического распределения памяти при программировании на C. Они есть:

1. malloc()
2. calloc()
3. free()
4. realloc()

Функция malloc

Метод **«malloc»** или **«memory allocation»** (выделение памяти) в C используется для динамического выделения одного большого блока памяти указанного размера. Он возвращает указатель типа void, который можно преобразовать в указатель любой формы. Он не инициализирует память во время выполнения, поэтому изначально инициализирует каждый блок значением мусора по умолчанию.

Синтаксис malloc() в C

```
ptr = (тип приведения*) malloc(размер в байтах)
```

Например:

```
ptr = (int*) malloc(100 * sizeof(int));
```

Поскольку размер int составляет 4 байта, этот оператор выделит 400 байт памяти. А указатель ptr содержит адрес первого байта в выделенной памяти.

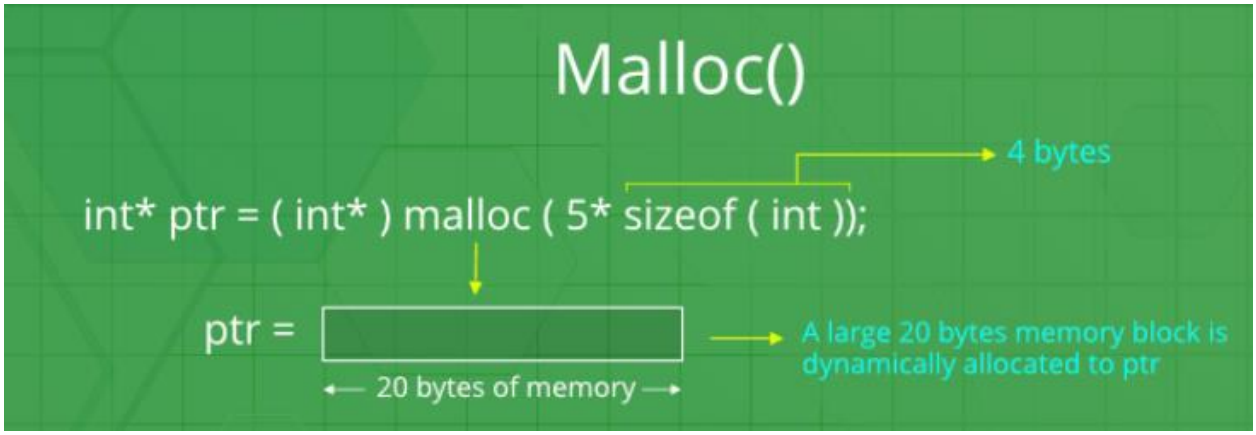

Рис. 2. Функция malloc

Если места недостаточно, выделение завершается неудачно и возвращается NULL-указатель.

Пример использования функции malloc

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

int main()
{

    // This pointer will hold the
    // base address of the block created
    int* ptr;
    int n, i;

    // Get the number of elements for the array
    printf("Enter number of elements:");
```

```
    scanf("%d",&n);
    printf("Entered number of elements: %d\n", n);

    // Dynamically allocate memory using malloc()
    ptr = (int*)malloc(n * sizeof(int));

    // Check if the memory has been successfully
    // allocated by malloc or not
    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {

        // Memory has been successfully allocated
        printf("Memory successfully allocated using malloc.\n");

        // Get the elements of the array
        for (i = 0; i < n; ++i) {
            ptr[i] = i + 1;
        }

        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }
    }

    return 0;
}
```

Выход:

```
Enter number of elements: 5
Memory successfully allocated using malloc.
The elements of the array are: 1, 2, 3, 4, 5,
```

Функция calloc

Метод «calloc» или **«contiguous allocation» (непрерывное выделение)** в C используется для динамического выделения указанного количества блоков памяти указанного типа. он очень похож на malloc(), но имеет два разных момента:

1. Он инициализирует каждый блок значением по умолчанию «0».
2. По сравнению с malloc(), он имеет два параметра или аргумента.

Синтаксис calloc() в C

```
ptr = (приведение типа*)calloc(n, размер элемента);
```

здесь n — нет. элементов, а element-size — это размер каждого элемента.

Например:

```
ptr = (float*) calloc(25, sizeof(float));
```

Этот оператор выделяет непрерывное пространство в памяти для 25 элементов, каждый из которых имеет размер числа с плавающей запятой.

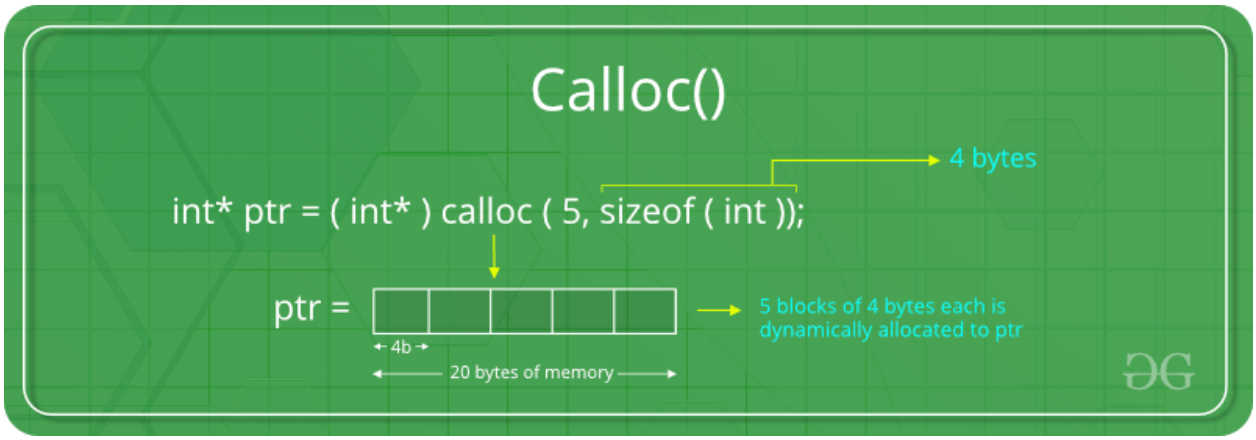

Рис. 3. Функция calloc

Если места недостаточно, выделение завершается неудачно и возвращается NULL-указатель.

Пример использования функции calloc

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

int main()
{
```

```
    // This pointer will hold the
    // base address of the block created
    int* ptr;
    int n, i;

    // Get the number of elements for the array
    n = 5;
    printf("Enter number of elements: %d\n", n);

    // Dynamically allocate memory using calloc()
    ptr = (int*)calloc(n, sizeof(int));

    // Check if the memory has been successfully
    // allocated by calloc or not
    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {

        // Memory has been successfully allocated
        printf("Memory successfully allocated using calloc.\n");

        // Get the elements of the array
        for (i = 0; i < n; ++i) {
            ptr[i] = i + 1;
        }

        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }
    }

    return 0;
}
```

Вывод:

```
Enter number of elements: 5
Memory successfully allocated using calloc.
The elements of the array are: 1, 2, 3, 4, 5,
```

Функция free

Метод **«free»** в C используется для динамического освобождения памяти. Память, выделенная с помощью функций malloc() и calloc(), не освобождается сама по себе. Следовательно, метод free() используется всякий раз, когда происходит динамическое выделение памяти. Это помогает уменьшить потери памяти, освобождая ее.

Синтаксис free() в C

```
free(ptr);
```

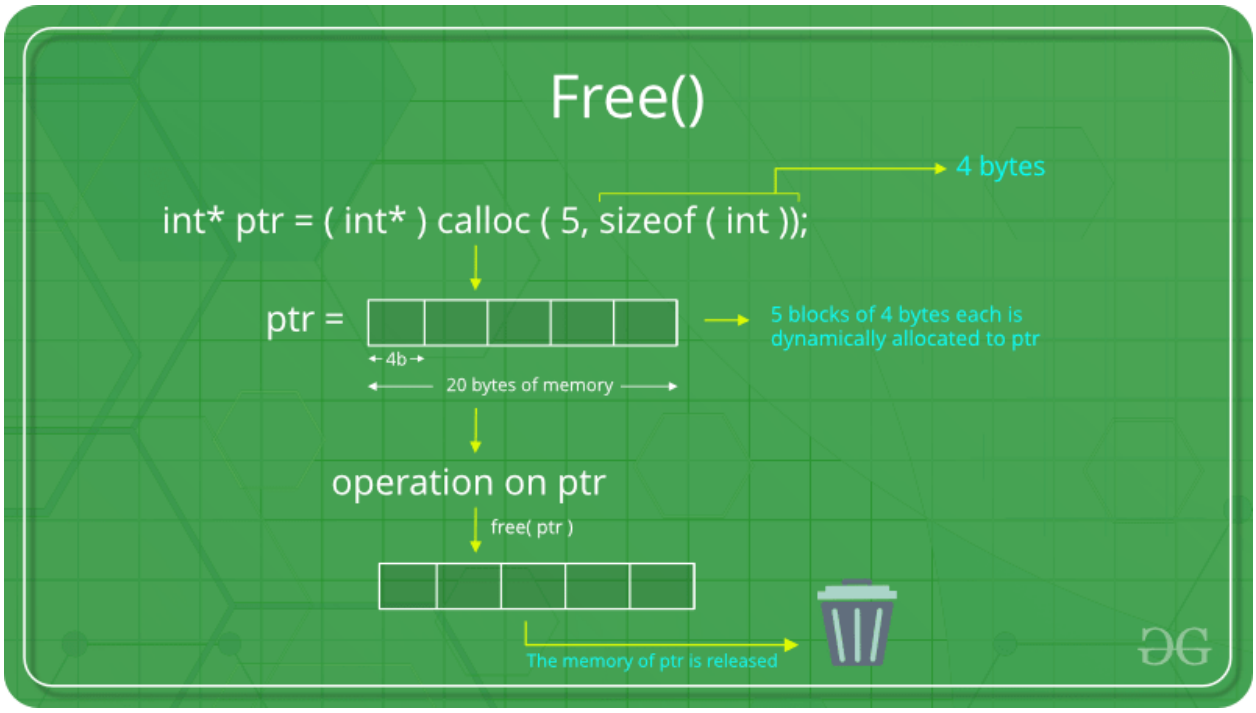

Рис. 4. Функция free

Пример использования функции free

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

int main()
{

    // This pointer will hold the
    // base address of the block created
    int *ptr, *ptr1;
    int n, i;

    // Get the number of elements for the array
```

```
    n = 5;
    printf("Enter number of elements: %d\n", n);

    // Dynamically allocate memory using malloc()
    ptr = (int*)malloc(n * sizeof(int));

    // Dynamically allocate memory using calloc()
    ptr1 = (int*)calloc(n, sizeof(int));

    // Check if the memory has been successfully
    // allocated by malloc or not
    if (ptr == NULL || ptr1 == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {

        // Memory has been successfully allocated
        printf("Memory successfully allocated using malloc.\n");

        // Free the memory
        free(ptr);
        printf("Malloc Memory successfully freed.\n");

        // Memory has been successfully allocated
        printf("\nMemory successfully allocated using calloc.\n");

        // Free the memory
        free(ptr1);
        printf("Calloc Memory successfully freed.\n");
    }

    return 0;
}
```

Вывод:

```
Enter number of elements: 5
Memory successfully allocated using malloc.
Malloc Memory successfully freed.

Memory successfully allocated using calloc.
Calloc Memory successfully freed.
```

Функция realloc

Метод «realloc» или **«re-allocation» (перераспределения)** в C используется для динамического изменения распределения ранее выделенной памяти. Другими словами, если памяти, ранее выделенной с помощью malloc или calloc, недостаточно, для **динамического перераспределения памяти** можно использовать realloc.

Перераспределение памяти сохраняет уже существующее значение, а новые блоки будут инициализироваться со значением мусора по умолчанию.

Синтаксис realloc() в C

```
ptr = realloc (ptr, newSize);
```

где ptr перераспределяется с новым размером «newSize».

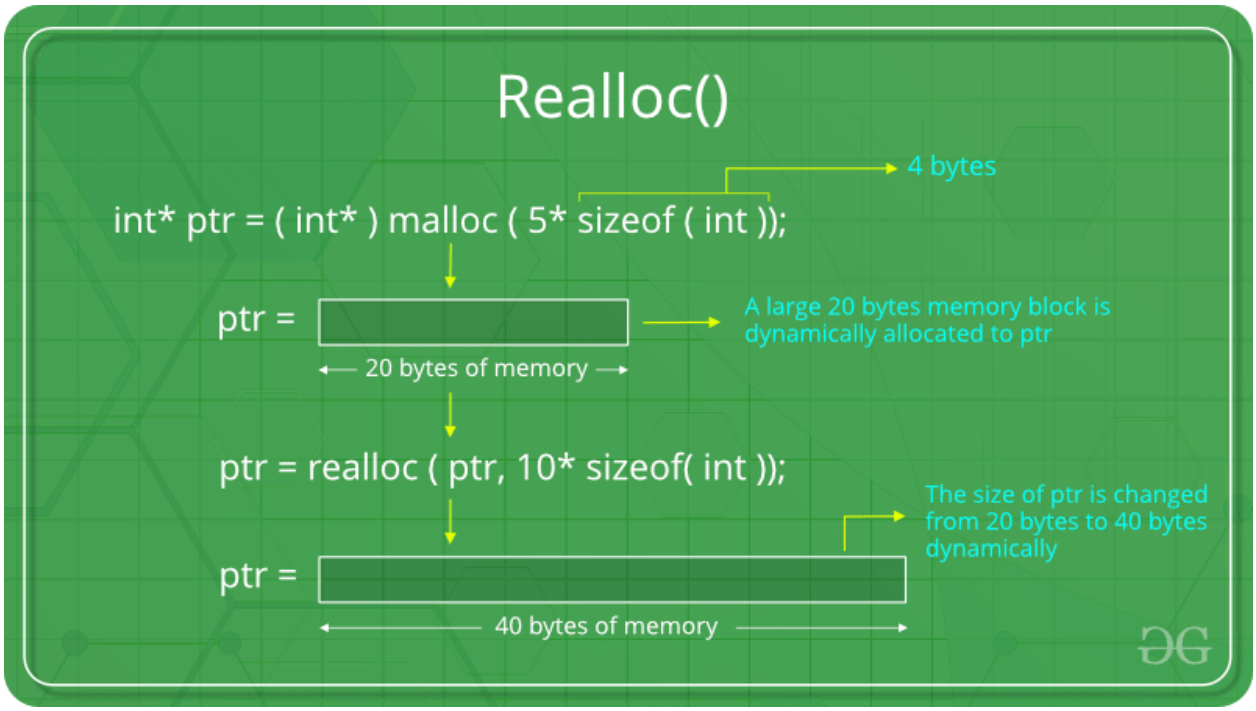

Рис. 5. Функция realloc

Если места недостаточно, выделение завершается неудачно и возвращается NULL-указатель.

Пример использования функции realloc

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

int main()
{
```

```
// This pointer will hold the
// base address of the block created
int* ptr;
int n, i;

// Get the number of elements for the array
n = 5;
printf("Enter number of elements: %d\n", n);

// Dynamically allocate memory using calloc()
ptr = (int*)calloc(n, sizeof(int));

// Check if the memory has been successfully
// allocated by malloc or not
if (ptr == NULL) {
    printf("Memory not allocated.\n");
    exit(0);
}
else {

    // Memory has been successfully allocated
    printf("Memory successfully allocated using calloc.\n");

    // Get the elements of the array
    for (i = 0; i < n; ++i) {
        ptr[i] = i + 1;
    }

    // Print the elements of the array
    printf("The elements of the array are: ");
    for (i = 0; i < n; ++i) {
        printf("%d, ", ptr[i]);
    }

    // Get the new size for the array
    n = 10;
    printf("\n\nEnter the new size of the array: %d\n", n);

    // Dynamically re-allocate memory using realloc()
    ptr = (int*)realloc(ptr, n * sizeof(int));
```

```
        // Memory has been successfully allocated
        printf("Memory successfully re-allocated using realloc.\n");

        // Get the new elements of the array
        for (i = 5; i < n; ++i) {
            ptr[i] = i + 1;
        }

        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }

        free(ptr);
    }

    return 0;
}
```

Вывод:

```
Enter number of elements: 5
Memory successfully allocated using calloc.
The elements of the array are: 1, 2, 3, 4, 5,

Enter the new size of the array: 10
Memory successfully re-allocated using realloc.
The elements of the array are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

Еще один пример метода realloc():

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int index = 0, i = 0, n,
        *marks; // this marks pointer hold the base address
                // of  the block created
    int ans;
    marks = (int*)malloc(sizeof(
        int)); // dynamically allocate memory using malloc
    // check if the memory is successfully allocated by
    // malloc or not?
```

```
if (marks == NULL) {
    printf("memory cannot be allocated");
}
else {
    // memory has successfully allocated
    printf("Memory has been successfully allocated by "
           "using malloc\n");
    printf("\n marks = %pc\n",
           marks); // print the base or beginning
                   // address of allocated memory
    do {
        printf("\n Enter Marks\n");
        scanf("%d", &marks[index]); // Get the marks
        printf("would you like to add more(1/0): ");
        scanf("%d", &ans);

        if (ans == 1) {
            index++;
            marks = (int*)realloc(
                marks,
                (index + 1)
                    * sizeof(
                        int)); // Dynamically reallocate
                               // memory by using realloc
            // check if the memory is successfully
            // allocated by realloc or not?
            if (marks == NULL) {
                printf("memory cannot be allocated");
            }
            else {
                printf("Memory has been successfully "
                       "reallocated using realloc:\n");
                printf(
                    "\n base address of marks are:%pc",
                    marks); ////print the base or
                            ///beginning address of
                            ///allocated memory
            }
        }
    } while (ans == 1);
    // print the marks of the students
    for (i = 0; i <= index; i++) {
```

```
            printf("marks of students %d are: %d\n ", i,
                    marks[i]);
        }
        free(marks);
    }
    return 0;
}
```

Вывод:

```
Memory has been successfully allocated by using malloc

 marks = 0x22eb010c

 Enter Marks:89
would you like to add more(1/0): 1
Memory has been successfully reallocated using realloc:

 base address of marks are:0x22eb010c
 Enter Marks:78
would you like to add more(1/0): 1
Memory has been successfully reallocated using realloc:

 base address of marks are:0x22eb010c
 Enter Marks:84
would you like to add more(1/0): 0
marks of students 0 are: 89
 marks of students 1 are: 78
 marks of students 2 are: 84
```

Рис. 6. Результат работы программы

Источники

- Dynamic Memory Allocation in C using malloc(), calloc(), free() and realloc()
- C Dynamic Memory Allocation
- Dynamic Memory Allocation in C

Задание на лабораторную работу

Для задачи, выданной на 1-ой лабораторной работе, написать 3 программы. Использовать функции динамического распределения памяти malloc, calloc, realloc, free.

1. Дана строка символов. Необходимо ввести ее, используя функцию malloc, free. Программа должна работать след. образом: для ввода 1-ого символа захватываем память с помощью функции malloc. Если строка не закончена, то захватываем память под два символа с помощью функции malloc, в нее самостоятельно переписываем введенные символы и добавляем один новый. Память, захваченную ранее, освобождаем с помощью функции free. И так делаем до тех пор, пока не введем всю строку. При этом каждый раз увеличиваем размер захватываемой памяти на один символ. Далее строка обрабатывается в соответствии с задачей.
2. Дана строка символов. Необходимо ввести ее, используя функцию calloc, realloc, free. Программа должна работать след. образом: для ввода нескольких символов (не более 5) захватываем память с помощью функции calloc. Если строка не закончена, то изменяем размер памяти с помощью функции realloc. При этом каждый раз захватываем память больше на выбранную нами добавляемую порцию (не более 5). И так делаем до тех пор, пока не введем всю строку. Далее строка обрабатывается в соответствии с задачей.
3. Строку вводим по 1-му или 2-му варианту, записываем ее в файл. Далее читаем строку из файла, выделяя память под нее как в 1-ом или 2-ом варианте, обрабатываем ее, результат помещаем в файл.