Управление динамическим выделением памяти (*stdlib.h*)

`void* malloc( size_t size );`
Выделяет блок памяти размером в указанное количество байт и возвращает указатель на его. Содержание выделенного блока памяти не инициализируется, оно остается с неопределенными значениями. В случае неудачного выполнения возвращает **`NULL`**.

`void* calloc( size_t num, size_t size );`
Выделяет память под указанное количество элементов с учетом их размера и возвращает указатель на начало выделенного блока. Весь блок инициализируется нулями. В случае неудачного выполнения возвращает **`NULL`**.

`void *realloc( void *ptr, size_t new_size );`
Изменяет размер ранее выделенного блока памяти, на начало которого ссылается указатель, до размера в указанное количество байт. Если указатель имеет значение **`NULL`**, то есть память не выделялась, то действие функции аналогично действию **`malloc`**.

`void free( void* ptr );`
Освобождает ранее выделенный блок памяти, на начало которого ссылается указатель.

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

int main(void) {
  setlocale(LC_ALL, "RU");

  int *ptr;

  ptr = malloc(10000);

  if (ptr == NULL) {
    puts("Ошибка при выделении памяти.");
  }

  return EXIT_SUCCESS;
}
```

```c
if ((ptr = malloc(10000)) == NULL)
if (!ptr)
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>

char* ConcatinateString(const char *firstStrting, const char *secondStrting) {
  char *result;

  result = malloc(strlen(firstStrting) + strlen(secondStrting) + 1);

  if (result == NULL) {
    puts("Ошибка: malloc в ConcatinateString");
    exit(EXIT_FAILURE);
  }

  strcpy(result, firstStrting);
  strcat(result, secondStrting);

  return result;
}

int main(void) {
  setlocale(LC_ALL, "RU");

  char *ptr;

  ptr = ConcatinateString("абв", "где");

  puts(ptr);

  return EXIT_SUCCESS;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>

int main(void) {
  setlocale(LC_ALL, "RU");

  int *array, n = 10;

  array = malloc(sizeof(int) * n);
  // calloc(n, sizeof(int));
  // realloc(array, sizeof(int) * m);

  for (int i = 0; i < n; i++) {
    printf("%d\n", array[i]);
  }

  return EXIT_SUCCESS;
}
```

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

enum MENU { quit };

typedef struct {
  int lowerBound, upperBound;
} Bound;

int GetInt(void);

size_t GetSize(const char *message);

void AllocateMatrix(double ***array, size_t rows, size_t columns);

double** AllocateMatrixAnotherWay(size_t rows, size_t columns);

double GetRandomNumber(Bound limits);

void FillMatrix(double **array, size_t rows, size_t columns);

void PrintMatrix(double **array, size_t rows, size_t columns);

void MultiplyMatrices(double **arrayA, size_t rowsA, size_t columnsA,
                      double **arrayB, size_t columnsB, double **result);

void FreeMatrix(double ***array, size_t rows);
```

```
int main(void) {
  setlocale(LC_ALL, "RU");
  srand(time(NULL));

  int userChoice = 0;
  size_t numberOfRowsInA = 0, numberOfColumnsInA = 0,
         numberOfRowsInB = 0, numberOfColumnsInB = 0;
  double **matrixA = NULL, **matrixB = NULL, **matrixC = NULL;

  do {
    numberOfRowsInA = GetSize("Количество строк матрицы A: ");
    numberOfColumnsInA = GetSize("Количество столбцов матрицы A (строк матрицы B): ");
    numberOfRowsInB = numberOfColumnsInA;
    numberOfColumnsInB = GetSize("Количество столбцов матрицы B: ");
    AllocateMatrix(&matrixA, numberOfRowsInA, numberOfColumnsInA);
    matrixB = AllocateMatrixAnotherWay(numberOfRowsInB, numberOfColumnsInB);
    FillMatrix(matrixA, numberOfRowsInA, numberOfColumnsInA);
    FillMatrix(matrixB, numberOfRowsInB, numberOfColumnsInB);
    puts("\nМатрица A:");
    PrintMatrix(matrixA, numberOfRowsInA, numberOfColumnsInA);
    puts("Матрица B:");
    PrintMatrix(matrixB, numberOfRowsInB, numberOfColumnsInB);
    AllocateMatrix(&matrixC, numberOfRowsInA, numberOfColumnsInB);
    MultiplyMatrices(matrixA, numberOfRowsInA, numberOfColumnsInA,
                     matrixB, numberOfColumnsInB, matrixC);
    puts("Матрица C:");
    PrintMatrix(matrixC, numberOfRowsInA, numberOfColumnsInB);
    FreeMatrix(&matrixA, numberOfRowsInA);
    FreeMatrix(&matrixB, numberOfRowsInB);
    FreeMatrix(&matrixC, numberOfRowsInA);

    printf("0 - Завершить работу.\n");
    userChoice = GetInt();
  } while (userChoice != quit);

  return EXIT_SUCCESS;
}
```

```
int GetInt(void) {
  int input = 0;
  while (!scanf("%d", &input)) {
    while (getchar() != '\n')
      ;
    printf("Ошибка ввода. Введите число.\n");
  }
  while (getchar() != '\n')
    ;
  return input;
}

size_t GetSize(const char *message) {
  int size = 0;
  do {
    printf("%s", message);
    size = GetInt();
  } while (size <= 0);
  return (size_t)size;
}

void AllocateMatrix(double ***array, size_t rows, size_t columns) {
  *array = (double**)calloc(rows, sizeof(double*));
  if (*array == NULL) {
    printf("Ошибка при выделении памяти.\n");
    exit(EXIT_FAILURE);
  }
  for(size_t i = 0; i < rows; ++i) {
    (*array)[i] = (double*)calloc(columns, sizeof(double));
    if ((*array)[i] == NULL) {
      printf("Ошибка при выделении памяти.\n");
      exit(EXIT_FAILURE);
    }
  }
}
```

```
double** AllocateMatrixAnotherWay(size_t rows, size_t columns) {
  double **array = (double**)calloc(rows, sizeof(double*));
  if (array == NULL) {
    printf("Ошибка при выделении памяти.\n");
    exit(EXIT_FAILURE);
  }
  for(size_t i = 0; i < rows; ++i) {
    array[i] = (double*)calloc(columns, sizeof(double));
    if (array[i] == NULL) {
      printf("Ошибка при выделении памяти.\n");
      exit(EXIT_FAILURE);
    }
  }
  return array;
}

double GetRandomNumber(Bound limits) {
  int fraction = 100;
  return (int)((limits.lowerBound
                + (double)rand() / RAND_MAX
                * (limits.upperBound - limits.lowerBound)) * fraction)
         / (double)fraction;
}

void FillMatrix(double **array, size_t rows, size_t columns) {
  Bound limits = { -100, 100 };
  for (size_t i = 0; i < rows; ++i) {
    for (size_t j = 0; j < columns; ++j) {
      array[i][j] = GetRandomNumber(limits);
    }
  }
}
```

```
void PrintMatrix(double **array, size_t rows, size_t columns) {
  puts("");
  for (size_t i = 0; i < rows; ++i) {
    for (size_t j = 0; j < columns; ++j) {
      printf("%-16.4f ", array[i][j]);
    }
    puts("");
  }
  puts("");
}

void MultiplyMatrices(double **arrayA, size_t rowsA, size_t columnsA,
                      double **arrayB, size_t columnsB, double **result) {
  for (size_t i = 0; i < rowsA; ++i) {
    for (size_t j = 0; j < columnsB; ++j) {
      for (size_t k = 0; k < columnsA; ++k) {
        result[i][j] += arrayA[i][k] * arrayB[k][j];
      }
    }
  }
}

void FreeMatrix(double ***array, size_t rows) {
  for (size_t i = 0; i < rows; ++i) {
    free((*array)[i]);
    (*array)[i] = NULL;
  }
  free(*array);
  *array = NULL;
}
```