Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Факультет информационных технологий и управления
Кафедра информационных технологий автоматизированных систем

Отчет по лабораторной работе № 3
«Адресная арифметика и управление памятью»
Вариант 8

Выполнил студент группы 120602 Анашкевич П. С. _____
Проверил ассистент Гончаревич А. Л. _____

Минск 2014

1 ЦЕЛЬ РАБОТЫ

Цели данной лабораторной работы:

– знакомство с указателями в языке Си;

– ознакомление с функциями работы с памятью языка Си и Windows API;

– приобретение практических навыков управления памятью в программах.

2 ХОД РАБОТЫ

2.1 Текст задания

Необходимо написать три функции, которые будут вызываться из функции main():

– первая функция получает размерность массива, создает динамический массив и возвращает указатель на начало созданного массива;

– вторая – получает адрес массива и его размерность и решает одну из ниже перечисленных задач;

– третья функция получает адрес массива и его размерность и освобождает память, занятую массивом.

Размерность вводится с клавиатуры в функции main() и передается в первую функцию. Значения элементов вводятся с клавиатуры. Для управления памятью использовать соответствующие функции Win32 API.

2.2 Теоретические сведения

Под управлением памятью имеются в виду возможности программы по размещению и манипулированию данными. Поскольку единственным «представителем» памяти в программе выступают переменные, то управление памятью определяется тем, каким образом работает с ними и с образованными ими структурами данных язык программирования.

Большинство языков программирования однозначно закрепляет за переменными их типы данных и ограничивает работу с памятью только теми областями, в которых эти переменные размещены. Программист не может выйти за пределы самим же определенного шаблона структуры данных. С другой стороны, это позволяет транслятору обнаруживать допущенные ошибки как в процессе трансляции, так и в процессе выполнения программы.

В языке Си ситуация принципиально иная по двум причинам. Во-первых, наличие операции адресной арифметики при работе с указателями позволяет, в принципе, выйти за пределы памяти, выделенной транслятором под переменную и адресовать память как «до» так и «после» нее. Другое дело, что это должно производиться осознанно и корректно. Во-

вторых, присваивание и преобразование указателей различных типов, речь о котором пойдет ниже, позволяет рассматривать одну и ту же память «под различным углом зрения» в смысле типов заполняющих ее переменных:

– Двумерный массив всегда реализован как одномерный массив с количеством элементов, соответствующих первому индексу, причем каждый элемент представляет собой массив элементов базового типа с количеством, соответствующим второму индексу.

– Идентификатор массива без скобок интерпретируется как адрес нулевого элемента нулевой строки, или указатель на базовый тип данных.

– Имя двумерного массива с единственным индексом интерпретируется как начальный адрес соответствующего внутреннего одномерного массива. A[i] понимается как &A[i][0], то есть начальный адрес i-го массива символов.

Сказанное справедливо и для N-мерных массивов: многомерный массив представляет собой массив элементов первого индекса, каждый из который представляет собой массив элементов второго индекса и т.д.

Для работы с динамической памятью в WinAPI предусмотрены функции, изображенные на рисунке 1.

```
void * GlobalAlloc(uiFlags, dwSize);
void * GlobalReAlloc(p, dwSize, uiFlags);
GlobalSize(void *);
GlobalFree(void *);
```

Рисунок 1 – Функции WinAPI для работы с памятью

За исключением одной, для каждой функции, начинающейся со слова Global, существует другая, начинающаяся со слова Local. Эти два набора функций в Windows идентичны. Два различных слова сохранены для совместимости с предыдущими версиями Windows, где функции Global возвращали дальние указатели, а функции Local – ближние.

2.3 Особенности разработанной программы

Разработанная программа имеет следующие функциональные особенности:

– интерактивный консольный интерфейс, устойчивый к ошибкам переполнения;

– осуществление проверки корректности вводимых данных с возможностью замены некорректных данных;

– разделение различных функциональных частей исходного кода программы по различным файлам с исходным кодом;

– реализована поддержка инкрементальной компиляции.

Для захвата памяти, необходимой для размещения двумерного массива, была разработана функция, представленная на рисунке 2.

```
int ** create_array(int N, int M)
{
  int i; const int uiFlags = 0; int ** a;

  a = (int **)GlobalAlloc(uiFlags, sizeof(int *) * N);
  if (a)
    {
      for (i = 0; i < N; i++)
        {
          a[i] = (int *)GlobalAlloc(uiFlags, sizeof(int) * M);
          if (!(a[i]))
            {
              for (i = i-1; i >=0; i--)
                GlobalFree(a[i]);
              GlobalFree(a);
              return NULL;
            }
        }
      return a;
    }
  else
    return NULL;
}
```

Рисунок 2 – Функция создания двумерного динамического массива

Функция поиска наименьшего элемента в двумерном массиве была использована функция, представленная на рисунке 3.

```
int
array_get_min(int ** array, int N, int M)
{
  int i, j, min = array[0][0];
  for (i = 0; i < N; i++)
    {
      for (j = 0; j < M; j++)
        {
          if (array[i][j] < min)
            min = array[i][j];
        }
    }
  return min;
}
```

Рисунок 3 – Функция поиска минимального элемента в массиве

Для освобождения памяти, отведенной под размещение двумерного массива, была разработана функция, представленная на рисунке 4.

```
void destroy_array(int ** array, int N, int M)
{
  int i;
  for (i = 0; i < N; i++)
    {
      GlobalFree(array[i]);
    }

  GlobalFree(array);
  array = NULL;
}
```

Рисунок 4 – Функция освобождения памяти

Полный исходный текст разработанной программы расположен в приложении А.

ЗАКЛЮЧЕНИЕ

В ходе проведения лабораторной работы мы познакомились с функциями работы с памятью в языке Си, ознакомились с указателями, приобрели практические навыки управления памятью в программах.

ПРИЛОЖЕНИЕ А
(справочное)
Программа поиска минимального элемента в массиве
(к подразделу 2.3)

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

int **
create_array(int N, int M)
/* Create NxM array in heap */
/* Return pointer to allocated memory,
   if succeed, NULL otherwise */
{
  int i;
  const int uiFlags = 0;
  int ** a;

  a = (int **)GlobalAlloc(uiFlags, sizeof(int *) * N);
  if (a)
    {
      for (i = 0; i < N; i++)
        {
          a[i] = (int *)GlobalAlloc(uiFlags, sizeof(int) * M);
          if (!(a[i]))
            {
              for (i = i-1; i >=0; i--)
                GlobalFree(a[i]);
              GlobalFree(a);
              return NULL;
            }
        }
      return a;
    }
  else
    return NULL;
}

void
print_array(int ** array, int N, int M)
```

```
/* Display array size and content */
{
  int i, j;
  puts("=== ARRAY DIMENSIONS ===");
  printf("N: %i; M: %i\n", N, M);
  puts("===== ARRAY CONTENTS =====");
  for (i = 0; i < N; i++)
    {
      for (j = 0; j < M; j++)
        {
          printf("%d ", array[i][j]);
        }
      printf("\n");
    }
  puts("==========================");
}

int
array_get_min(int ** array, int N, int M)
/* Display array size and content */
{
  int i, j, min = array[0][0];
  for (i = 0; i < N; i++)
    {
      for (j = 0; j < M; j++)
        {
          if (array[i][j] < min)
            min = array[i][j];
        }
    }
  return min;
}

void
destroy_array(int ** array, int N, int M)
/* Free array's allocated memory */
{
  int i;
  for (i = 0; i < N; i++)
    {
      GlobalFree(array[i]);
    }

  GlobalFree(array);
```

```
  array = NULL;
}

static void
clear_input()
{
  while (getchar() != '\n');
}

static void
print_usage()
{
  const char * usage_msg =
    "Press \'s\' to start calculations, \'q\' to exit";
  puts(usage_msg);
}

static int
get_dimension(int * dest)
/* Get array dimension from user input */
/* Return 1 if correct, 0 otherwise */
{
  if (!scanf("%i", dest) || (*dest <= 0))
    return 0;
  return 1;
}

int
get_options(struct io_opts * dest_opts)
/* Get user input from interactive console interface */
/* Return 0 if need to exit */
{
  char choice;
  print_usage();

  while (1)
    {
      scanf("%c", &choice);
      clear_input();
      switch (choice)
        {
        case 's':
        case 'S':
          {
```

```
            int N, M;

            puts("Input first dimension size:");
            while (!get_dimension(&N))
              {
                puts("Size of dimension must be greater than zero"
    );
                clear_input();
              }
            dest_opts->N = N;
            clear_input();

            puts("Input second dimension size:");
            while (!get_dimension(&M))
              {
                puts("Size of dimension must be greater than zero"
    );
                clear_input();
              }
            dest_opts->M = M;
            clear_input();

            return 1;
          }
        case 'q':
        case 'Q':
          return 0;
        default:
          {
            print_usage();
            break;
          }
        }
    }
}

static int
get_value(int i, int j)
/* Get [i,j] array value from user input */
{
  int val = 0;
  printf("Input A[%d][%d]\n", i, j);

  while (!scanf("%i", &val))
```

```
    {
      puts("Incorrect input!");
      clear_input();
    }
    clear_input();

    return val;
}


void
get_array_contents(int ** dest_array, int N, int M)
/* Fill array with user input */
{
  int i, j;

  for (i = 0; i < N; i++)
    for (j = 0; j < M; j++)
      {
        dest_array[i][j] = get_value(i, j);
      }
}

int main()
{
  struct io_opts options;
  while (get_options(&options))
    {
      int ** a;
      int min = 0;

      if (a = create_array(options.N, options.M))
        {
          get_array_contents(a, options.N, options.M);

          print_array(a, options.N, options.M);

          min = array_get_min(a, options.N, options.M);
          printf("Min value: %d\n", min);

          destroy_array(a, options.N, options.M);
        }
      else
        {
```

```
            puts("Error of memory allocation!");
        }
    }
  return 0;
}
```