ЛАБОРАТОРНАЯ РАБОТА 1.
РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ В WINDOWS

Цель работы: создание приложения с использованием WinAPI, демонстрирующего управление динамической памятью различными способами, а также реализация пользовательской кучи для динамического распределения памяти.

Основные теоретические сведения

1.1 Управление динамической памятью

Управление памятью в Windows может быть выполнено с использованием различных функций и API операционной системы. Рассмотрим несколько примеров кода на языке C/C++ для выделения и освобождения памяти в Windows.

Выделение памяти с использованием `malloc` и `free` (C/C++):

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

int main() {
    // Выделение памяти под массив целых чисел
    int *arr = (int*)malloc(5 * sizeof(int));

    if (arr == NULL) {
        printf("Не удалось выделить память\n");
        return 1;
    }

    // Использование выделенной памяти
    for (int i = 0; i < 5; i++) {
        arr[i] = i * 10;
    }

    // Освобождение памяти после использования
    free(arr);

    return 0;
}
```

Выделение памяти с использованием функции VirtualAlloc (WinAPI):

```c
#include <Windows.h>
#include <stdio.h>

int main() {
```

```
    // Выделение 1 мегабайта (1048576 байт) виртуальной памяти
    LPVOID mem = VirtualAlloc(NULL, 1048576, MEM_COMMIT, PAGE_READWRITE);

    if (mem == NULL) {
        printf("Не удалось выделить виртуальную память\n");
        return 1;
    }

    // Использование выделенной виртуальной памяти

    // Освобождение виртуальной памяти
    VirtualFree(mem, 0, MEM_RELEASE);

    return 0;
}
```

Выделение и освобождение памяти с использованием C++ операторов new и delete:

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

int main() {
    SetConsoleOutputCP(1251);
    // Выделение памяти под одно целое число
    int *num = new int;

    // Использование выделенной памяти
    *num = 42;
    std::cout << "Значение: " << *num << std::endl;

    // Освобождение памяти
    delete num;

    return 0;
}
```

1.2 Функции для работы с кучей

WinAPI предоставляет ряд функций для работы с кучей (памятью, выделяемой в куче). Рассмотрим эти функции более подробно:

- **HeapCreate** – создает новую кучу.

Синтаксис:

```
HANDLE HeapCreate(DWORD flOptions, SIZE_T dwInitialSize, SIZE_T dwMaximumSize);
```

Пример:

```
HANDLE hHeap = HeapCreate(0, 0, 0);
```

- **HeapAlloc** – выделяет блок памяти из кучи.

Синтаксис:

```
LPVOID HeapAlloc(HANDLE hHeap, DWORD dwFlags, SIZE_T dwBytes);
```

Пример:

```
int* pData = (int*)HeapAlloc(hHeap, 0, sizeof(int) * 10);
```

- **HeapFree** – освобождает блок памяти, выделенный ранее с помощью `HeapAlloc`.

Синтаксис:

```
BOOL HeapFree(HANDLE hHeap, DWORD dwFlags, LPVOID lpMem);
```

Пример:

```
HeapFree(hHeap, 0, pData);
```

- **HeapReAlloc** – изменяет размер выделенного блока памяти в куче.

Синтаксис:

```
LPVOID HeapReAlloc(HANDLE hHeap, DWORD dwFlags, LPVOID lpMem, SIZE_T dwBytes);
```

Пример:

```
pData = (int*)HeapReAlloc(hHeap, 0, pData, sizeof(int) * 20);
```

- **HeapDestroy** – уничтожает кучу и освобождает все связанные с ней ресурсы.

Синтаксис:

```
BOOL HeapDestroy(HANDLE hHeap);
```

Пример:

```
HeapDestroy(hHeap);
```

- **HeapSize** – возвращает размер выделенного блока памяти в куче.

Синтаксис:

```
SIZE_T HeapSize(HANDLE hHeap, DWORD dwFlags, LPCVOID lpMem);
```

Пример:

```
SIZE_T size = HeapSize(hHeap, 0, pData);
```

- **HeapValidate** – проверяет целостность кучи и выделенных блоков.

Синтаксис:

```
BOOL HeapValidate(HANDLE hHeap, DWORD dwFlags, LPCVOID lpMem);
```

Пример:

```
if (HeapValidate(hHeap, 0, pData)) {
    printf("Куча валидна.\n");
} else {
    printf("Куча повреждена.\n");
}
```

Пример 1. Создание кучи и выделение памяти:

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

int main() {
    SetConsoleOutputCP(1251);
    // Создание кучи
    HANDLE hHeap = HeapCreate(0, 0, 0);

    if (hHeap == NULL) {
        printf("Не удалось создать кучу\n");
        return 1;
    }

    // Выделение памяти из кучи
    int *data = (int*)HeapAlloc(hHeap, 0, sizeof(int) * 5);

    if (data == NULL) {
        printf("Не удалось выделить память из кучи\n");
        HeapDestroy(hHeap);
        return 1;
    }

    // Использование выделенной памяти
    for (int i = 0; i < 5; i++) {
        data[i] = i * 10;
    }

    // Освобождение памяти
    HeapFree(hHeap, 0, data);

    // Уничтожение кучи
    HeapDestroy(hHeap);

    return 0;
}
```

Пример 2. Выделение строки в куче:

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

int main() {
    SetConsoleOutputCP(1251);
    // Создание кучи
    HANDLE hHeap = HeapCreate(0, 0, 0);

    if (hHeap == NULL) {
        printf("Не удалось создать кучу\n");
        return 1;
    }

    // Выделение строки в куче
```

```
    char *str = (char*)HeapAlloc(hHeap, 0, 256);

    if (str == NULL) {
        printf("Не удалось выделить память для строки\n");
        HeapDestroy(hHeap);
        return 1;
    }

    // Копирование строки в выделенную память
    strcpy_s(str, 256, "Пример строки в куче");

    // Использование строки

    // Освобождение памяти
    HeapFree(hHeap, 0, str);

    // Уничтожение кучи
    HeapDestroy(hHeap);

    return 0;
}
```

Порядок выполнения работы

Задание 1. Выполнить задание согласно варианту, используя различные способы выделения и освобождения динамической памяти:

- использование `malloc`, `realloc`, `free`;
- использование `new`, `delete` (только на C++);
- использование `VirtualAlloc`, `VirtualFree`;
- использование `HeapAlloc`, `HeapFree`.

Варианты задания:

1. Создать динамический массив. Размер вводится пользователем. Заполнить массив случайными числами. Вывести на экран содержимое массива и сумму чисел в массиве. Прибавить сумму чисел к каждому элементу массива, вывести массив на экран. Освободить память.
2. Создать динамический массив. Размер вводится пользователем. Заполнить массив случайными числами. Вывести на экран содержимое массива и среднее значение чисел в массиве. Прибавить среднее значение к каждому элементу массива, вывести массив на экран. Освободить память.
3. Создать динамический массив. Размер вводится пользователем. Заполнить массив случайными числами. Вывести на экран содержимое массива и максимальное число в массиве. Прибавить максимальное значение к каждому элементу массива, вывести массив на экран. Освободить память.
4. Создать динамический массив. Размер вводится пользователем. Заполнить массив случайными числами. Вывести на экран содержимое массива и минимальное число в массиве. Прибавить минимальное значение к каждому элементу массива, вывести массив на экран. Освободить память.

5. Создать динамический массив. Размер вводится пользователем. Заполнить массив случайными числами. Вывести на экран содержимое массива и количество четных чисел в массиве. Прибавить это количество к каждому элементу массива, вывести массив на экран. Освободить память.

6. Создать динамический массив. Размер вводится пользователем. Заполнить массив случайными числами. Вывести на экран содержимое массива и количество нечетных чисел в массиве. Прибавить это количество к каждому элементу массива, вывести массив на экран. Освободить память.

7. Создать динамический массив. Размер вводится пользователем. Заполнить массив случайными числами. Вывести на экран исходное содержимое массива и отсортированное по возрастанию содержимое. Освободить память.

8. Создать динамический массив. Размер вводится пользователем. Заполнить массив случайными числами. Вывести на экран исходное содержимое массива и отсортированное по убыванию содержимое. Освободить память.

9. Создать динамический массив. Размер вводится пользователем. Заполнить массив случайными числами. Вывести на экран исходное содержимое массива, переставить числа в массиве в обратном порядке и вывести на экран. Освободить память.

10. Создать динамический массив. Размер вводится пользователем. Заполнить массив элементами ряда Фибоначчи. Вывести на экран исходное содержимое массива, переставить числа в массиве в обратном порядке и вывести на экран. Освободить память.

Задание 2. Выполнить с использованием пользовательской кучи (функции `HeapCreate`, `HeapAlloc`, `HeapReAlloc`, `HeapFree`, `HeapDestroy`):

Создать пользовательскую кучу. Создать динамический массив из указателей. Размер вводится пользователем. Для каждого элемента массива выделить память по отдельности из пользовательской кучи. Указатели на созданные элементы сохранить в массиве. Выполнить задание с этим массивом согласно варианту из задания 1. В конце освободить память, уничтожив пользовательскую кучу.

Контрольные вопросы

1. *Основные концепции:*

- Что такое динамическая память и как она отличается от статической памяти?
- Какие проблемы могут возникнуть при неправильном использовании динамической памяти?

2. *Функции управления памятью:*

- Какие функции Windows API используются для работы с динамической памятью?
- В чем различие между функциями `HeapAlloc` и `VirtualAlloc`?

- Как освободить память, выделенную функцией `HeapAlloc`? А функцией `VirtualAlloc`?

3. *Кучи (Heap):*
   - Что такое куча (heap) и как она используется для управления памятью?
   - Как создать и уничтожить кучу с помощью Windows API?
   - Какие параметры можно задать при создании кучи?
4. *Виртуальная память:*
   - Что такое виртуальная память и как она управляется в Windows?
   - Какие преимущества и недостатки есть у использования виртуальной памяти по сравнению с кучей?
5. *Отладка и диагностика:*
   - Каковы основные инструменты и методы отладки утечек памяти в Visual Studio?
   - Как обнаружить и исправить утечки памяти в программах?
6. *Оптимизация и эффективность:*
   - Какие методы можно использовать для минимизации фрагментации памяти?
   - Каковы основные принципы написания эффективного кода, работающего с динамической памятью?
7. *Практическое применение:*
   - Опишите процесс выделения и освобождения памяти с использованием функции `HeapAlloc`.
   - Приведите пример кода, использующего функции `VirtualAlloc` и `VirtualFree`.
   - Какие меры предосторожности необходимо соблюдать при работе с динамической памятью?
8. *Анализ и сравнение:*
   - В каких случаях предпочтительно использовать кучу, а в каких – виртуальную память?
   - Какое влияние на производительность оказывает использование кучи и виртуальной памяти?