ФИНАНСОВО ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ

КОЛЛЕДЖ КОСМИЧЕСКОГО МАШИНОСТРОЕНИЯ И ТЕХНОЛОГИИ

Курсовой проект

По дисциплине «Прикладное программирование»

Тема: **«Работа с памятью»**

Выполнил студент:

Моисеев Алексей Андреевич

№ группы: К26П

Подпись студента: \_\_\_\_\_\_\_\_\_\_

Принял: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Королев 2014

**Содержание**

[Введение 4](#_Toc386903146)

[1. Постановка задачи 4](#_Toc386903147)

[2. Класс для работы с памятью 4](#_Toc386903148)

[2.1. Описание класса 4](#_Toc386903149)

[2.2. Описание алгоритмов 5](#_Toc386903150)

[2.2.1. Метод «ProcessMemory::ValueToBytes» 5](#_Toc386903151)

[2.2.2. Метод «ProcessMemory::Search» 7](#_Toc386903152)

[2.2.3. Метод «ProcessMemory::Filter» 8](#_Toc386903153)

[3. Базовый класс для создания окон 9](#_Toc386903154)

[4. Функция «WinMain» 10](#_Toc386903155)

[4.1. Общие сведения. 10](#_Toc386903156)

[4.2. Исходный код. 10](#_Toc386903157)

[4.3. Блок-схема 11](#_Toc386903158)

[5. Описание программы 12](#_Toc386903159)

[5.1. Общие сведения. 12](#_Toc386903160)

[5.2. Функциональное назначение. 12](#_Toc386903161)

[5.3. Используемые технические средства. 12](#_Toc386903162)

[5.4. Вызов и загрузка. 12](#_Toc386903163)

[6. Графический интерфейс 13](#_Toc386903164)

[6.1. Главное окно 13](#_Toc386903165)

[6.2. Окно «Поиск» 15](#_Toc386903166)

[6.3. Окно «Отсев» 16](#_Toc386903167)

[6.4. Окно «О программе» 17](#_Toc386903168)

[6.5. Окно «Добавить адрес» 18](#_Toc386903169)

[6.6. Окно «Изменение типа» 18](#_Toc386903170)

[6.7. Окно изменение значения, описания и адреса 19](#_Toc386903171)

[7. Руководство пользователя 20](#_Toc386903172)

[7.1. Поиск точного значения 20](#_Toc386903173)

[7.2. Поиск неизвестного значения 24](#_Toc386903174)

[7.3. Добавление и удаление адресов в главной таблице 31](#_Toc386903175)

[7.4. Изменение значения 34](#_Toc386903176)

[7.5. Заморозка значения 35](#_Toc386903177)

[Заключение 36](#_Toc386903178)

[Список литературы 37](#_Toc386903179)

# Введение

В данном курсовом проекте описано построение и реализация приложения для Windows, позволяющее изменять значение в памяти программы (например, в играх можно сделать очень много денег, патронов, здоровье и так далее). Изменять значение в памяти можно во всех программах и практически во всех играх. Также в программе можно заморозить значение (например, в играх можно заморозить здоровье и тогда у вас никогда не закончится здоровье).

# Постановка задачи

Цель работы – написать программу, которая будет читать байты в оперативной памяти, искать в них определенные значения и редактировать эти значения. Для этого нужно научиться читать и записывать байты в оперативную память, научиться работать с графическим интерфейсом.

# Класс для работы с памятью

# Описание класса

В классе «ProcessMemory» следующие методы:

1. «SetProcess» - Записываем в переменную hProcess HANDLE процесса.
2. «Size» - Количество найденных адресов.
3. «Search» - Поиск значения в памяти процесса.
4. «Filter» - Отсеивание адресов.
5. «GetProcess» - HANDLE процесса.
6. «CloseHandleProcessMemory» - Закрывает HANDLE процесса.
7. «OpenProc» - Открываем процесс для работы с ним.
8. «ValueToBytes» - Переводит значения в байты.

Этот класс нужен для того, что бы осуществлять поиск значения в памяти процесса и для отсеивания не нужных адресов. Обращаться к адресам можно с помощью «[]» (Например, std::cout << ProcMem[4] << std::endl; Таким образом мы выведем в консоль 5-ый адрес, который нашли в памяти).

WinApi функции для работы с памятью:

1. «ReadProcessMemory» - Чтение байтов из памяти процесса.
2. «WriteProcessMemory» - Запись байтов в память процесса.
3. «memcpy» - Копирование байт.
4. «VirtualQueryEx» -Узнаёт информацию о регионе памяти (размер, базовый адрес, защиту).

Функции которые понадобятся для класса:

* 1. «StrtoValue<type>» - Переводит из строки в type.

# Описание алгоритмов

# Метод «ProcessMemory::ValueToBytes»

byte \*ProcessMemory::ValueToBytes(const char \*Value, const int &Type, size\_t &SizeBytes);

Этот метод нужен для того, что бы переводить значение из строки в байты.

В строке «Value» хранится само значение, которое нужно перевести, в значение «Type» записан тип значения (Byte, 2 Byte, 4 Byte, 8 Byte, Double, Float, Text и Array of Bytes), в переменную «SizeBytes» нужно записать количество байт которые нам понадобятся.

Значение переменной «Type»:

* + 1. Если «Type» = 0, то значит что у нас тип «Byte». Записываем в переменную «SizeBytes» значение «1», создаём динамический массив размером «SizeBytes» байт и назовём его «Bytes» и создадим переменную типа «int» и назовём эту переменную «buffer». В переменную «Buffer» записываем значение функции StrtoValue<int>(Value), потом воспользуемся WinApi функцией memcpy(Bytes, &buffer, SizeBytes).
    2. Если «Type» = 1, то значит что у нас тип «2 Byte». Записываем в переменную «SizeBytes» значение «2», создаём динамический массив размером «SizeBytes» байт и назовём его «Bytes» и создадим переменную типа «int» и назовём эту переменную «buffer». В переменную «Buffer» записываем значение функции StrtoValue<int>(Value), потом воспользуемся WinApi функцией memcpy(Bytes, &buffer, SizeBytes).
    3. Если «Type» = 2, то значит что у нас тип «4 Byte». Записываем в переменную «SizeBytes» значение «4», создаём динамический массив размером «SizeBytes» байт и назовём его «Bytes» и создадим переменную типа «int» и назовём эту переменную «buffer». В переменную «Buffer» записываем значение функции StrtoValue<int>(Value), потом воспользуемся WinApi функцией memcpy(Bytes, &buffer, SizeBytes).
    4. Если «Type» = 3, то значит что у нас тип «8 Byte». Записываем в переменную «SizeBytes» значение «8», создаём динамический массив размером «SizeBytes» байт и назовём его «Bytes» и создадим переменную типа «\_\_int64» и назовём эту переменную «buffer». В переменную «Buffer» записываем значение функции StrtoValue<\_\_int64>(Value), потом воспользуемся WinApi функцией memcpy(Bytes, &buffer, SizeBytes).
    5. Если «Type» = 4, то значит что у нас тип «Double». Записываем в переменную «SizeBytes» значение функции (sizeof(double)), создаём динамический массив размером «SizeBytes» байт и назовём его «Bytes» и создадим переменную типа «double» и назовём эту переменную «buffer». В переменную «Buffer» записываем значение функции StrtoValue<double>(Value), потом проверяем - есть ли в строке «Value» точка, если её там нет, то значение в переменной «Buffer» округляем. Потом воспользуемся WinApi функцией memcpy(Bytes, &buffer, SizeBytes).
    6. Если «Type» = 5, то значит что у нас тип «Float». Записываем в переменную «SizeBytes» значение функции (sizeof(float)), создаём динамический массив размером «SizeBytes» байт и назовём его «Bytes» и создадим переменную типа «float» и назовём эту переменную «buffer». В переменную «Buffer» записываем значение функции StrtoValue<float>(Value), потом проверяем - есть ли в строке «Value» точка, если её там нет, то значение в переменной «Buffer» округляем. Потом воспользуемся WinApi функцией memcpy(Bytes, &buffer, SizeBytes).
    7. Если «Type» = 6, то значит что у нас тип «Text». Записываем в переменную «SizeBytes» значение функции (strlen(Value)), создаём динамический массив размером «SizeBytes» байт и назовём его «Bytes». Потом воспользуемся WinApi функцией memcpy(Bytes, Value, SizeBytes).
    8. Если «Type» = 7, то значит что у нас тип «Array of Bytes». Удаляем из строки «Value» все пробелы, если значение функции (strlen(Value)) не чётное, то в переменную «Value» добавляем в предпоследнюю позицию символ «0». Потом записываем в переменную «SizeBytes» значение функции (strlen(Value) / 2), создаём динамический массив размером «SizeBytes» байт и назовём его «Bytes». Потом каждые два символа из строки «Value» переводим в байт WinApi функцией «sscanf» и записываем в массив «Bytes».

Потом завершаем метод, возвращая массив байтов «Bytes».

# Метод «ProcessMemory::Search»

void ProcessMemory::Search(const bool &TypeSearch, const int &Type, const char \*Value);

Этот метод нужен для того, что бы искать значения в памяти процесса.

Значение «TypeSarch» - это тип поиска (известный или неизвестный), значение «Type» - это тип переменной, строка «Value» - в этой строке записано значение.

Проверяем, подключены ли мы к процессу, если нет, то завершаем метод. Если мы подключены к процессу, то освобождаем память от предыдущего поиска (очищаем вектор «MemoryRegions»). Потом перебираем все регионы памяти, используем функцию «VirtualQueryEx» что бы узнать какая стоит защита у этого региона, если защита стоит «PAGE\_READWRITE» (доступная для чтения и записи), то добавляем этот регион памяти в вектор «MemoreRegions», если стоит другая защита, то переходим к следующему региону памяти. Далее создаём динамический массив байт и называем эту переменную «Bytes» и создаём переменную типа «size\_t» и называем «SizeBytes» переводим строку «Value» в байты с помощью метода «ValueToButes(Value, Type, SizeBytes)» и записываем результат в переменную «Bytes».

Если «TypeSearch» равен «true» значит, мы ищем неизвестное значение и поэтому мы сохраняем все адреса.

Если «TypeSearch» равен «false», то мы ищем известное значение. Для этого мы обращаемся к каждому элементу вектора «MemoryRegions» и вызываем метод «GetBytesMemoryProcess» (этот метод возвращает указатель на массив байт) и ищем в этих байтах, байты которые нам нужны (сравнивать байты мы будем с помощью WinApi функцией «memcmp»).

Потом ещё раз перебираем все элементы вектора «MemoryRegions» и вызываем метод «empty» (этим методом мы проверяем, есть ли в этом регионе памяти адреса, если адресов нет, то возвращаем «true»). Если этот метод выдал «true», то удаляем этот элемент из вектора.

# Метод «ProcessMemory::Filter»

void ProcessMemory::Filter(const bool &TypeSearch, const int &Deistvie, const int &Type, const char \*Value);

Этот метод нужен для того, что бы отсеивать ненужные адреса.

В переменной «TypeSearch» записан тип поиска (известное значение или неизвестное), в переменной «Deistvie» записано действие для неизвестного значения (значение изменилось, значение не изменилось, значение увеличилось, значение уменьшилось), в переменной «Type» записан тип переменной, которую будем искать (Byte, 2 Byte, 4 Byte, 8 Byte, Double, Float, Text, Array of Bytes), в строке «Value» записано значение, которое будем искать.

Сначала проверяем - подключены мы к процессу или нет, если нет, то завершаем процесс. Если подключены, то создаём динамический массив байт «Bytes» и переменную типом «size\_t» и назовём её «SizeBytes». Вызываем метод «ValueToBytes(Value, Type, SizeBytes)» и записываем результат в переменную «Bytes». Перебираем все элементы вектора «MemoryRegions» и создаём динамический массив байт размером «MemoryRegions[i].GtSizeMemoryProcess()» и называем его «buffer», а потом читаем из памяти процесса регион памяти и записываем в «buffer», читаем из памяти процесса WinApi функцией «ReadProcessMemory». Создаём переменную типом «Memory memory(MemoryRegions[i].GetBaseAddress(), MemoryRegions[i].GrtSizeMemoryProcess(), buffer);». Потом будем искать наши «Bytes» в байтах, которые мы прочитали из памяти процесса «buffer». Перебираем массив байт «buffer» и сравниваем, если переменная «TypesSearch» равна:

* + - 1. «true», то значит значение неизвестное и если «Deistvie» равно:
         1. 0, то ищём значение, которое изменилось.
         2. 1, то ищем значение, которое не изменилось.
         3. 2, то ищем значение, которое увеличилось.
         4. 3, то ищем значение, которое уменьшилось.
      2. «false», то значение известное, тогда просто ищем байты в «buffer», которые равны байтам «Bytes».

Для тех значений, которые совпали, вызываем метод у переменной «memory», который сохраняет адрес «memory.push\_back(MemoryRegions[i][j]);». Меняем элемент вектора «MemoryRegions» на переменную «memory» методом «MemoryRegions[i].assign(memory);». Потом переходим к следующему элементу вектора «MemoryRegions».

Потом заново перебираем элементы вектора и вызываем метод «empty»(Возвращает «true» если в этом регионе нет ни одного адреса) и если возвращает «true», то удаляем данный элемент из вектора.

# Базовый класс для создания окон

Этот класс, базовый класс для всех окон в этой программе.

В классе «Window» следующие методы:

1. «Create» - Метод для добавления кнопок и многое другое в текущее окно.
2. «Show» - Показывает или скрывает окно.
3. «ShowDialog» - Показывает текущее окно и блокирует родительское окно.
4. «Update» - Обновление окна.
5. «Enable» - Блокировка окна, можно включать и выключать.
6. «IsEnable» - Узнает- активно ли окно.
7. «Close» - Метод для закрытия окна.
8. «WndProc» - Метод для обработки сообщения окна. В этом методе обработаны сообщения:
   1. Закрытие главного и дочернего окна.
   2. Нажатие на кнопку «Отмена».
   3. Сообщения по умолчанию «DefWindowProc(hWnd, Msg, wParam, lParam)».

# Функция «WinMain»

# Общие сведения.

Функция «WinMain» вызывается системой как начальная точка входа. В этой функции мы создаём главное окно, получаем сообщения и обрабатываем эти сообщения.

# Исходный код.

//Вызывается системой как начальная точка входа

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE PrevInstance, PSTR scCmdSize, int nCmdShow)

{

MSG msg;//Переменна, в неё будет записываться сообщение

//Создаём главное окно

WindowMain WinMain("Работа с памятью", WS\_OVERLAPPEDWINDOW, 650, 400, NULL, hInstance);

WinMain.Create();

//Выводим на экран главное окно

WinMain.Show(nCmdShow);

//Обновляем главное окно

WinMain.Update();

//Получаем сообщений

while (GetMessage(&msg, NULL, 0, 0))

{

//Эта функция переводит virtual-key сообщения в символьные сообщения

TranslateMessage(&msg);

//Эта функция отправляет сообщения оконной процедуре (WndProc)

DispatchMessage(&msg);

}

return msg.wParam;

}

# Блок-схема

# C:\Users\USER\Desktop\Курсовой проэкт\WinMain.jpeg

Рис. 1 Блок-схема (WinMain)

# Описание программы

# Общие сведения.

Программа называется «Работа с памятью». Исполняемый файл называется «Работа с памятью.exe». Проект реализован на языке «C++», графический интерфейс реализован с помощью «WinApi» в среде «Microsoft Visual Studio 2010» под управлением операционной системы «Microsoft Windows 7». Для функционирования программы необходима установленная операционная система Microsoft Windows 2000 / XP / Vista / 7 / 8. Программа будет также функционировать и в операционной системе Linux, при установленном эмуляторе Microsoft Windows.

# Функциональное назначение.

Программа предназначена для изменения значения в памяти программы. Значение можно искать в памяти программы, изменять, замораживать. К значению можно обращаться по адресу. Тип значения (Byte, 2 Byte, 4 Byte, 8 Byte, Double, Float, Text, Array of Bytes).

# Используемые технические средства.

Для работы программы необходимы системный блок, носитель (жесткий диск, дискета, флеш – накопитель, CD или DVD диск), монитор и клавиатура. Характеристики должны быть достаточными для работы операционной среды «Microsoft Windows 2000/XP/Vista/7/8».

# Вызов и загрузка.

Для вызова программы, то есть исполняемого файла «Работа с памятью.exe», необходимо использовать правила запуска программ в среде Windows.

На рабочем столе необходимо перейти в папку, содержащую «Работа с памятью.exe» и нажать «Enter» на картинке файла или выполнить двойной щелчок. Можно создать ярлык программы на панели быстрого запуска, тогда «работа с памятью.exe» можно вызывать одним щелчком.

При работе в командной строке из каталога программы можно набрать на клавиатуре «Работа с памятью.exe» и нажать клавишу «Enter».

# Графический интерфейс

Графический интерфейс нарисован с помощью «WinApi».

Классы для окон:

* + - * 1. Window – базовый класс для окон.
        2. WindowMain – класс главного окна.
        3. WindowSearch – класс окна «Поиск».
        4. WindowFilter – класс окна «Отсев».
        5. WindowAddAddress – класс окна «Добавить адрес».
        6. WindowEditType – класс окна «Изменение типа».
        7. WindowEditValue – класс окна для изменения значения, адреса и описания.

# Главное окно

В главном окне 6 кнопок (Поиск, Отсев, О программе, Добавить адрес, Добавить элемент из таблицы поиска в главную таблицу, Добавить все элементы из таблицы поиска в главную таблицу), 2 таблицы (Таблица поиска и главная таблица) и 1 ComboBox (Список процессов). Главное окно показано на рисунках 2 – 4. На рисунке 2 подробно описано главное окно.

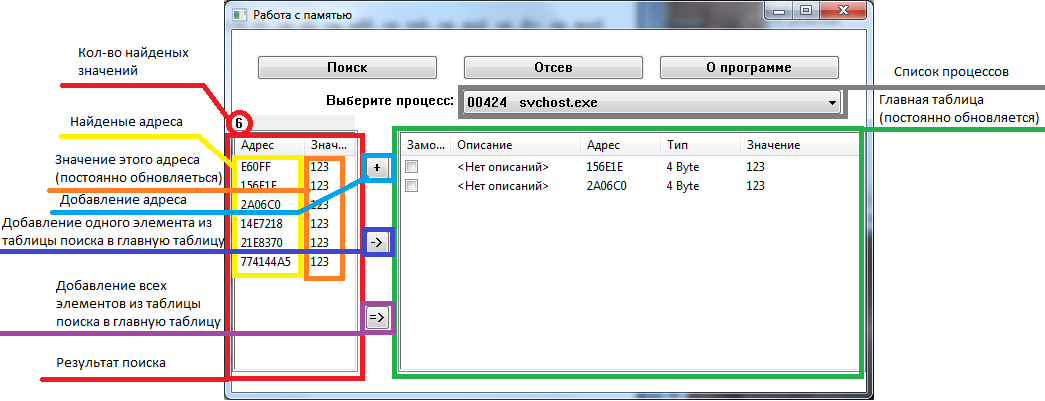


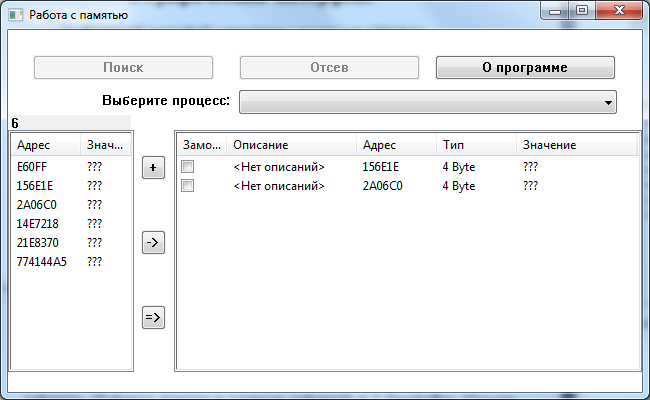
Рис. 2 Главное окно(1)

Рис. 3 Главное окно(2)

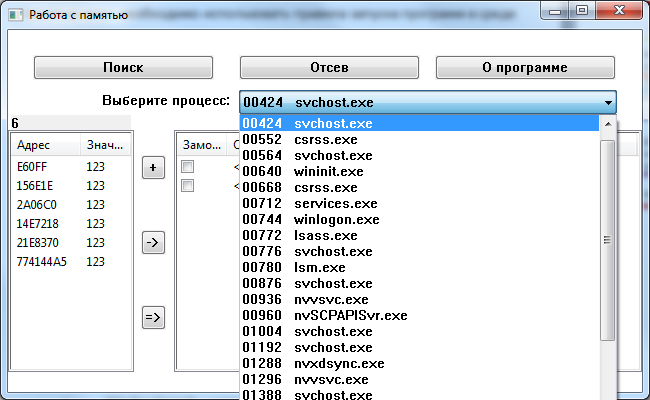


Рис. 4 Главное окно, список процессов

# Окно «Поиск»

Окно «Поиск» состоит из 2 ComboBox (тип поиска и тип значения), 2 кнопки (ок и отмена), 1 TextBox (значение). Окно поиска показано на рисунках 5 – 7.

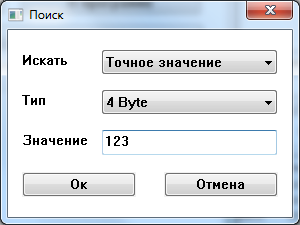


Рис. 5 Окно «Поиск»

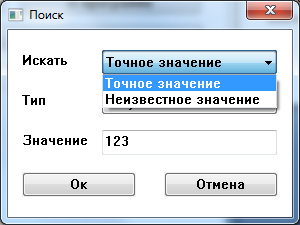


Рис. 6 окно «Поиск», тип поиска

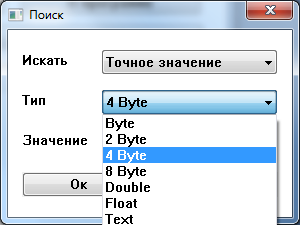


Рис. 7 окно «Поиск», тип значения

# Окно «Отсев»

Окно «Отсев» состоит из 2 ComboBox (тип поиска и действие для неизвестного значения), 2 кнопок (ок и отмена) и 1 TextBox (значение). Окно «Отсев» показано на рисунках 8 – 11.

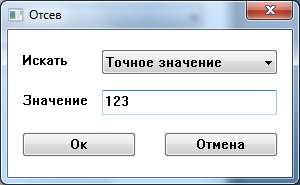


Рис. 8 окно «Отсев»

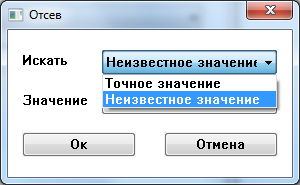


Рис. 9 окно «Отсев», тип поиска

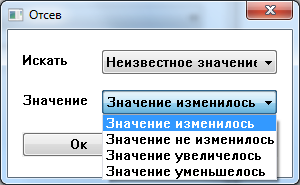


Рис. 10 окно «Отсев», тип значения

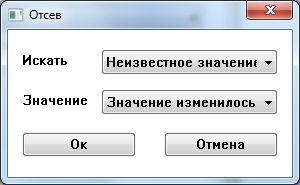


Рис. 11 окно «Отсев» (2)

# Окно «О программе»

Окно «О программе» имеет всего лишь одну кнопу (ок). Это окно показано на рисунке 12.

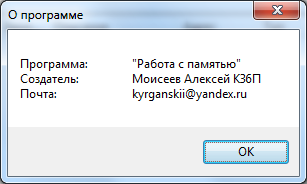


Рис. 12 окно «О программе»

# Окно «Добавить адрес»

В окне «Добавить адрес» 2 кнопки (ок и отмена), 3 TextBox (адрес, описание, размер текста) и 1 ComboBox (тип значения). Это оно показано на рисуке 13.

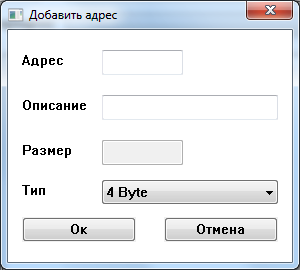


Рис. 13 окно «Добавить адрес»

# Окно «Изменение типа»

В окне «Изменение типа» 1 ComboBox (тип значения) и 2 кнопки (ок и отмена). Окно «Изменение типа» изображено на рисунках 14 – 15.

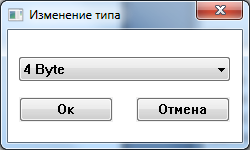


Рис. 14 окно «Изменение типа»

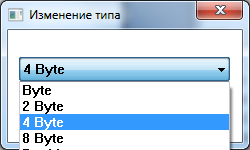


Рис. 15 окно «Изменение типа», тип значения

# Окно изменение значения, описания и адреса

В окнах изменение значения, описания и адреса всё одинаково - у них 1 TextBox (Значение) и 2 кнопки (ок и отмена). Окно «Изменение адреса» показано на рисунке 16, окно «Изменение описание» показано на рисунке 17 и окно «Изменение значения» показано на рисунке 18.

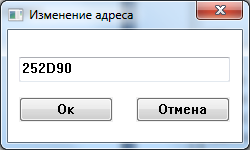


Рис. 16 окно «Изменение адреса»

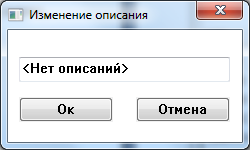


Рис. 17 окно «Изменение описания»

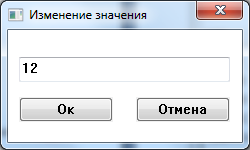


Рис. 18 окно «Изменение значения»

# Руководство пользователя

# Поиск точного значения

Сейчас я покажу как искать точное значение - я буду искать в игре «Thief» адрес значения денег. Запустите программу «Работа с памятью», запустите игру «Thief» и потом сверните игру, используя комбинацию клавиш «alt+tab» и переключитесь на программу «Работа с памятью». Там, где написано «выберите процесс», нажимаем и выбираем «Shipping-ThifeGame.exe».

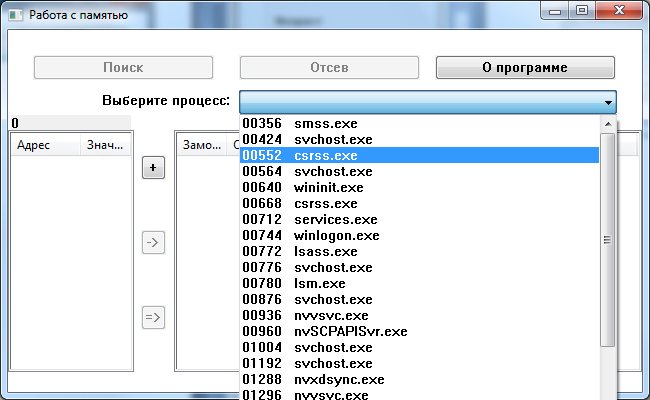


Рис. 19 Главное окно, список процессов

Потом переключаемся на игру и запоминаем сколько у нас денег. На рисунке 20 изображена игра «Thief».

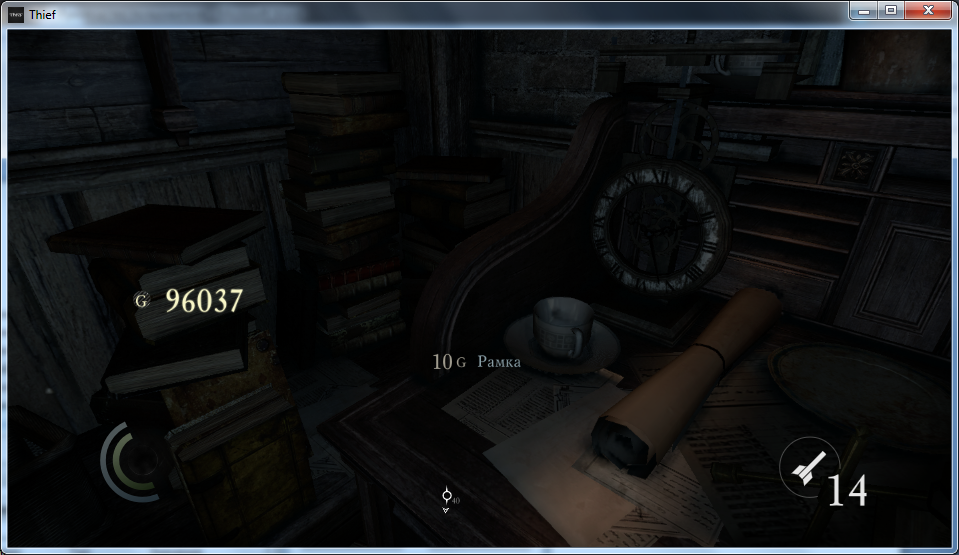


Рис. 20 игра «Thief», значения 1

У нас денег «96037», потом переключаемся на программу и нажимаем на кнопку «Поиск». В поле «Искать» выбираем «Точное значение», в поле «Тип» выбираем тип нашего значения (тип «Byte» значение от 0 до 255, тип «2 Byte» - от 0 до 65535, в основном все целые числа записаны в «4 Byte», там диапазон от -2147483648 до 2147483647, тип «8 Byte» самое большое число, тип «Float» - это значение с плавающей точкой, тип «Double» - это значение с плавающей точкой в двое точнее, чем «Float», тип «Text» - это строка, тип «Array of Bytes» - это массив байт) в данном случае тип нашего значения «4 Byte» и в поле «Значение» вводим значение, которое будем искать «96037» и нажимаем «Ок». На рисунке 21 изображено как нужно заполнить окно. Найденные адреса находятся в таблице поиска (рисунок 23).

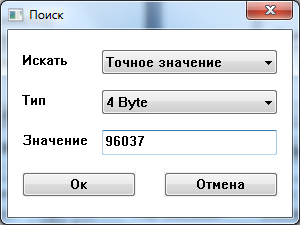


Рис. 21 окно «Поиск»

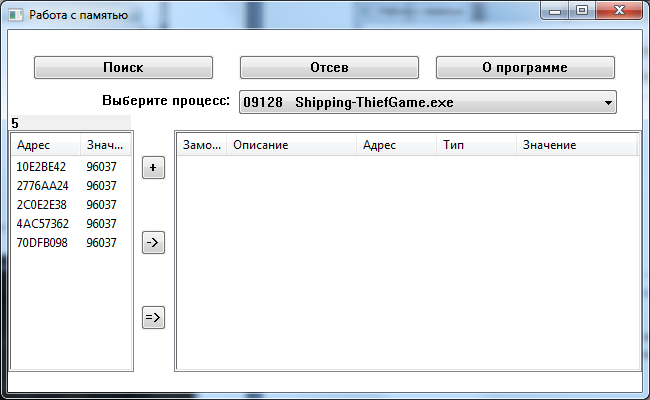


Рис. 22 Главное окно, таблица поиска

Мы нашли 5 адресов со значением «96037», что бы узнать какой из этих адресов отвечает за количество денег, нужно в игре ещё раз изменить значение (например, можно что – ни будь продать или купить).



Рис. 23 Игра «Thief» значение 2

У нас значение увеличилось на «3», теперь нам надо искать значение «96040». Нажимаем в программе кнопку «Отсев». В поле искать выбираем тип поиска, в данном случае нам известно, что значение увеличилось и значение равное «96040», можно сделать отсеивание двумя способами:

1. Тип поиска выбираем «Точное значение» и значение вводим «96040», как показано на рисунке 24.

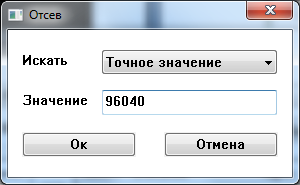


Рис. 24 окно «Отсев», точное значение

1. Тип поиска выбираем «Неизвестное значение» и выбираем значение «Значение увеличилось» , как показано на рисунке 25.

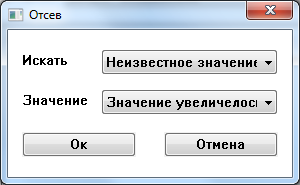


Рис. 25 окно «Отсев» неизвестное значение

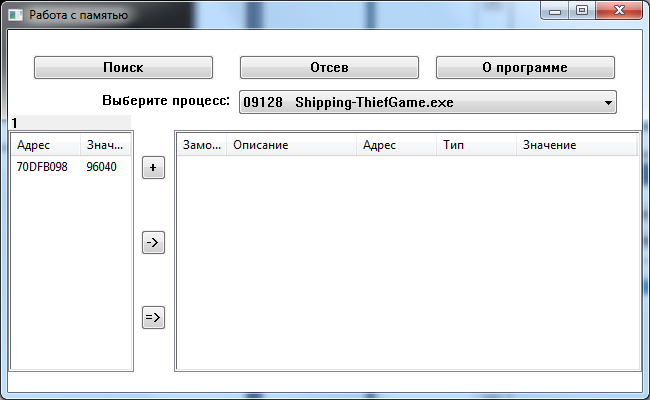


Рис. 26 Главное окно, таблица поиска(1)

На 26-ом рисунке показано что мы нашли 1 адрес. Мы нашли адрес, который отвечает за количество денег в игре.

# Поиск неизвестного значения

Сейчас я покажу как искать неизвестное значение, я буду искать в игре «Need For Speed Carbon» адрес значения, каторый отвечает за шкалу «Спидбрейк». Запустите программу «Работа с памятью», запустите «NFS Carbon», сверните игру alt+tab и переключитесь на программу «Работа с памятью». Там, где написано «выберите процесс» - нажимаем и «nfsc.exe».

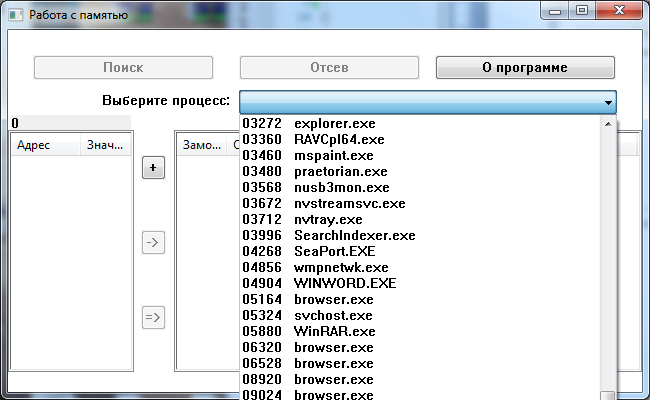


Рис. 26 Главное окно, список процессов

Потом нажимаем кнопку «Поиск», в поле «Искать» - выбираем «Неизвестное значение», в поле «Тип» выбираем «Float» (потому что чаще всего у значения, которое отвечает за шкалу тип «Float», так же координаты игрока x, y, z тоже «Float») и нажимаем кнопку «Ок». Заполнить окно «Поиск» нужно точно также как показано на рисунке 27. Окно может ненадолго зависнуть (из-за того что выполняется поиск).

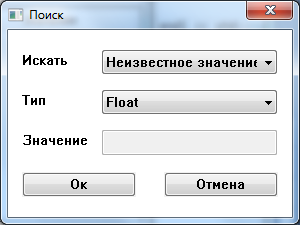


Рис. 27 Поиск неизвестного значения

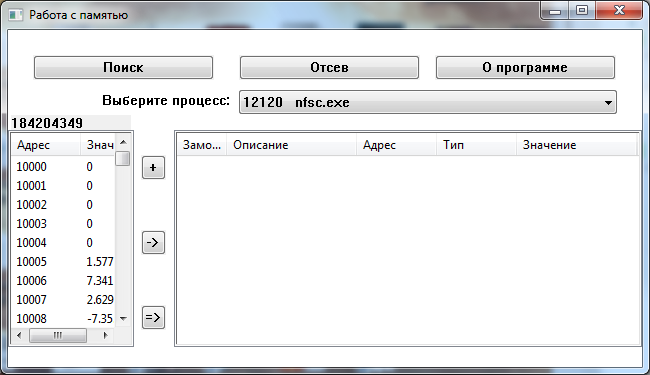


Рис. 198 Главное окно, таблица поиска

На рисунке 28 мы видем что нашлось 184204349 адресов, это очень много, что бы уменьшить количество адресов надо делать отсеивание, для этого надо изменить шкалу (я уменьшу шкалу «Спидбрейк»).



Рис. 29 Игра «Nfs» шкала(1)

Всё, шкалу мы уменьшили, теперь нажимаем кнопку «Отсев», в поле «Искать» - выбираем «Неизвестное значение», в поле «Значение» - выбираем «Значение уменьшилось» и жмём кнопку «Ок». Заполнить окно «Отсев» нужно точно также как показано на рисунке 30. Окно может ненадолго зависнуть (из-за того что выполняется отсеивания).

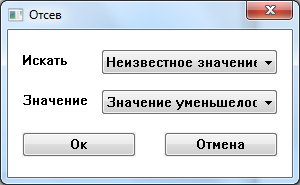


Рис. 30 окно «Отсев» значение уменьшилось

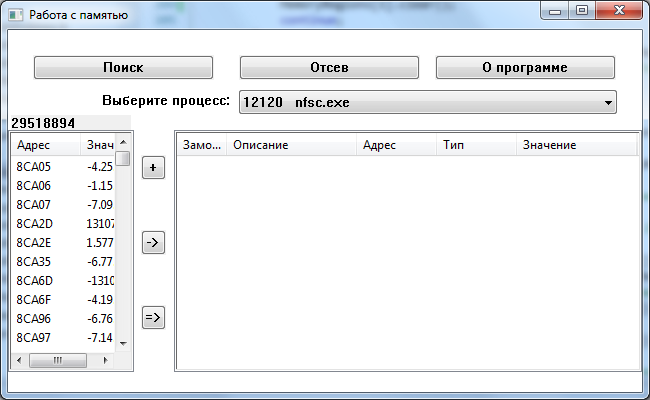


Рис. 31 Главное окно, таблица поиска

Теперь у нас адресов 29518894 (как показано на рисунке 31), делаем ещё раз отсеивание, я в игре шкалу увеличу.



Рис. 32 Игра «Nfs» шкала(2)

Нажимаем кнопку «Отсев», в поле «Искать» - выбираем «Неизвестное значение», в поле «Значение» - выбираем «Значение увеличилось» и жмём кнопку «Ок». Заполнить окно «Отсев» нужно точно также как показано на рисунке 33.

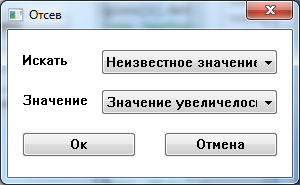


Рис. 33 окно «Отсев» значение увеличилось

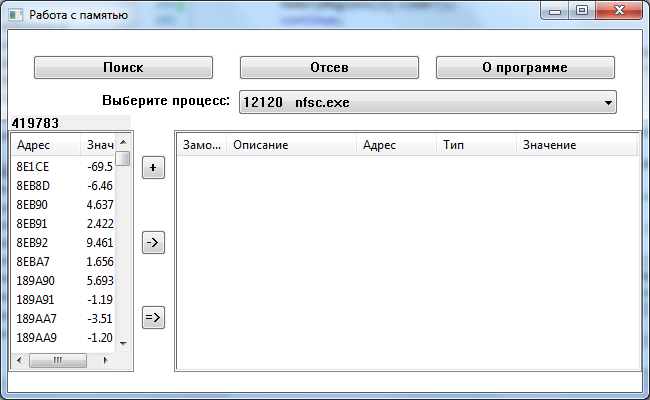


Рис. 34 Главное окно, таблица поиска

У нас теперь значений 419783 (как показано на рисунке 34) - это всё равно очень много, делаем ещё несколько раз отсеивание, пока не станет мало значений.

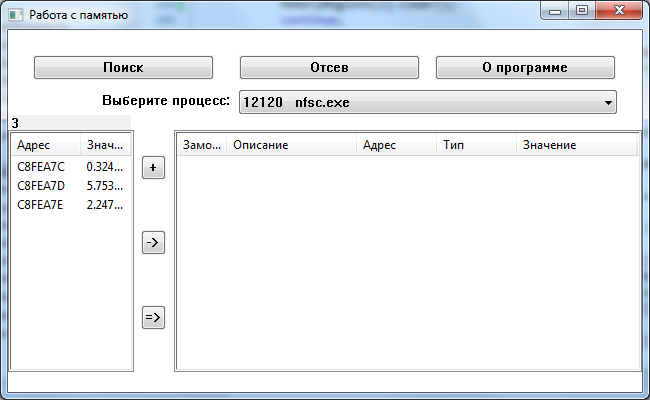


Рис. 35 Главное окно, таблица поиска

У нас теперь 3 адреса (рисунок 35) - это достаточно. Теперь изменяем значение в 1 адресе, проверяем изменилось ли значение в игре - если да, то это тот адрес, который отвечает за шкалу, если же не изменилась шкала, то меняем значение у следующего адреса.

# Добавление и удаление адресов в главной таблице

Добавлять адреса в таблицу можно 4-мя способами:

* + - 1. Нажать два раза на строку в таблице поиска (рисунок 36).

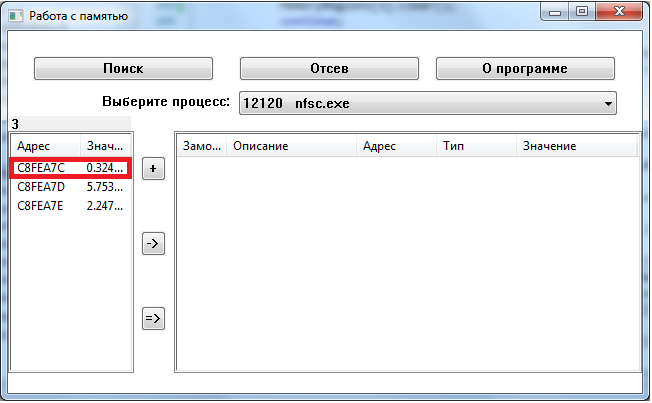


Рис. 36 Таблица поиска

* + - 1. Выделить элемент в таблице поиска и нажать на кнопку «->» (рисунок 37).

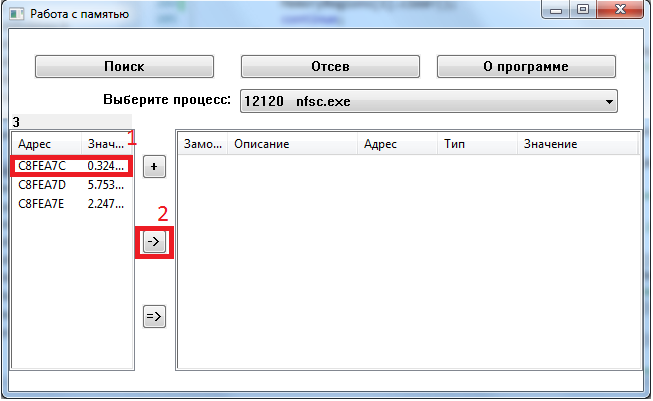


Рис. 37 Таблица поиска и кнопка «->»

* + - 1. Добавление всех элементов из таблицы поиска можно сделать, нажав на кнопку «=>» (рисунок 38).

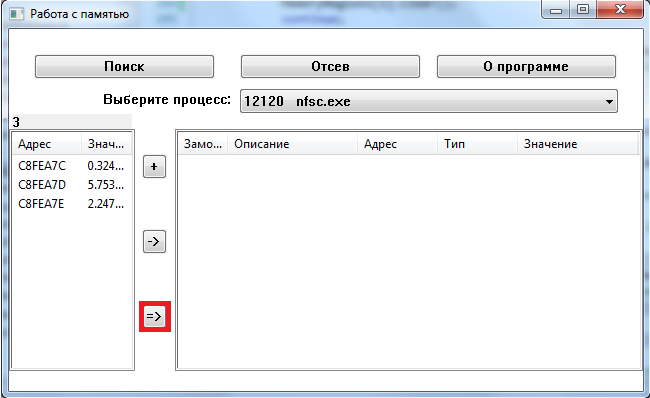


Рис. 38 кнопка «=>»

* + - 1. И последний способ добавления адреса в главную таблицу. Можно вручную ввести адрес, нажав на кнопку «+» (рисунок 39).

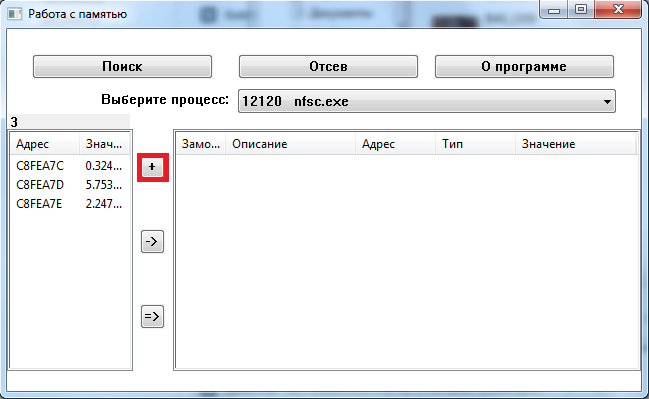


Рис. 39 кнопка «+»

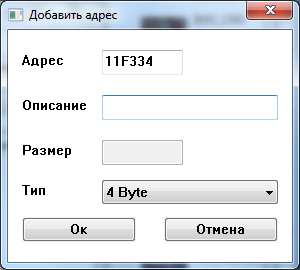


Рис. 40 окно «Добавить адрес»

Откроется окно «Добавить адрес» (рисунок 40). В поле адрес введите адрес, который надо добавить, в поле описание - можете вписать описание это адреса (например «здоровье»), но его не обязательно писать, в поле «Размер» - нужно вписывать значение, если у вас тип значения стоит «Text» или «Array of Bytes» и последнее поле «Тип» - в него указывайте тип вашего значения.

# Изменение значения

Изменять значение можно два раза нажав на ячейку в главной таблице. Те значения которые можно изменять обведены в прямоугольник на рисунке 41.

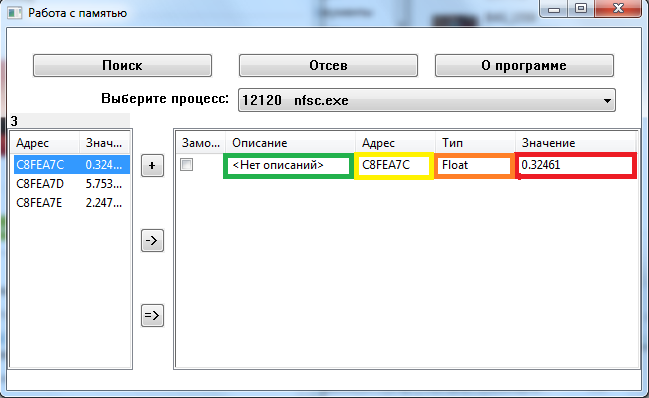


Рис. 41 Главная таблица

Два раза нажав на ячейку:

* + - 1. «Значение» - откроется окно «Изменение значения» (рисунок 42).

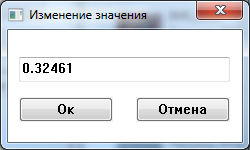


Рис. 42 Изменение значения

Введите нужное вам значение и нажмите кнопку «Ок».

* + - 1. «Тип» откроется окно «Изменение типа» (рисунок 43), выберите нужный вам тип и нажмите кнопку «Ок».

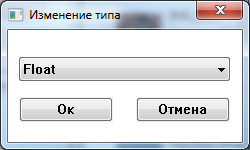


Рис. 43 Изменение типа

* + - 1. «Адрес» - откроется окно «Изменение адреса» (рисунок 44), введите нужный вам адрес и нажмите кнопку «Ок».

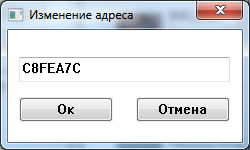


Рис. 44 Изменение адреса

* + - 1. «Описание» - откроется окно «Изменение описания» (рисунок 45), введите нужное описание и нажмите кнопку «Ок».

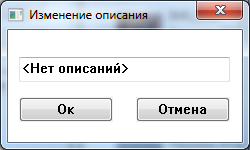


Рис. 45 Изменение описания

Что бы удалить адрес в главной таблице, нужно два раза нажать по этой строке ПКМ (правой кнопкой мыши).

# Заморозка значения

Что бы заморозить значение, нужно добавить адрес в главную таблицу и поставить галочку в ячейке «Заморозить» (На рисунке 46 обведено в прямоугольник), или выделить нужную строку в главной таблице и нажать пробел.

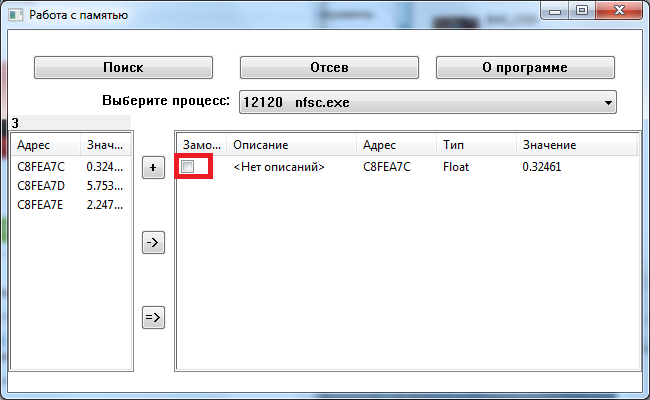


Рис. 46 Заморозка значения

# Заключение

В результате выполнения данной курсовой работы была разработана программа «Работа с памятью», которая ищет определенные значения в памяти процесса, изменяет эти значения и замораживает. В процессе работы было изучено: чтение и запись в память процесса, преобразование из значения в массив байт, работа с потоками и работа с «WinApi» (создание окон, таблиц, кнопок и так далее). В данной программе используется 10 классов, 1 структура и создаётся 3 дополнительных потока. В программе глобальные переменные не используются.

# Список литературы

1. Липпман С.Б. Язык программирования C++. Вводный курс, 4-е издание.
2. Ч. Петзольд. Программирование для Windows 95.
3. <http://msdn.microsoft.com/>
4. <http://www.youtube.com/user/coder1994/>