Министерство образования Республики Беларусь Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра Информатики

Дисциплина: СП

Отчет по курсовому проекту на тему

Деинсталлятор программ

Студент: гр.253504

Волчецкий А. М.

Руководитель:

ассистент кафедры информатики

Михневич М. В.

Минск 2015

Содержание

1. Введение………….………….………….………….………….…………...2
2. Выбор языка и технологий………….………….………….…………...…3

2.1 Описание языка программирования C#……………………………...3

2.2 Описание среды разработки Visual Studio 2015...…………………...5

2.3 Описание технологии WPF…………………………………………...6

1. Алгоритм работы приложениия………….…………………………….....8
2. Разработка приложения………….……………………………………….11
3. Описание основных функций приложения………….………………….12
4. Инструкция по использованию программного продукта…………........16
5. Заключение………….……………………………………………………..22

Список используемых источников………….…………………………………..23

1. **Введение**

Цель данного курсового проекта – разработка деинсталлятора программ. В нем должны быть реализованы такие функции, как:

1. Удаление установленных программ.
2. Удаление файлов и папок, оставшихся после удаления программы.
3. Удаление записей в реестре, оставшихся после удаления программы
4. Управление автозагрузкой.
5. **Выбор языка и технологий**

Для выполнения данного курсового проекта мною был выбран язык программирования С# и среда разработки Visual Studio 2015.

* 1. **Описание языка программирования C#**

На сегодняшний момент язык программирования C# один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программ до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

По сравнению с другими языками C# достаточно молодой, но в то же время он уже прошел большой путь. Первая версия языка вышла вместе с релизом Microsoft Visual Studio .NET в феврале 2002 года. Текущей версией языка является версия C# 6.0, которая вышла 20 июля 2015 года вместе с Visual Studio 2015.

C# является языком с Си-подобным синтаксисом. C# является объектно-ориентированным и в этом плане много перенял у Java и С++. Например, C# поддерживает полиморфизм, наследование, перегрузку операторов. Объектно-ориентированный подход позволяет решать задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И C# продолжает активно развиваться, и с каждой новой версией появляется все больше интересных функциональностей, как, например, лямбды, асинхронные методы и т.д.

*Роль платформы .NET*

Когда говорят C#, нередко имеют в виду технологии платформы .NET (WPF, ASP.NET). И, наоборот, когда говорят .NET, нередко имеют в виду C#. Однако, хотя эти понятия связаны, отождествлять их неверно. Язык C# был создан специально для работы с фреймворком .NET, однако само понятие .NET несколько шире.

Фреймворк .NET представляет мощную платформу для создания приложений. Можно выделить следующие ее основные черты:

*Поддержка нескольких языков*. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), благодаря чему .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#. При компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language) - своего рода ассемблер платформы .NET. Поэтому мы можем сделать отдельные модули одного приложения на отдельных языках.

*Кроссплатформенность*. .NET является переносимой платформой (с некоторыми ограничениями). Например, последняя версия платформы на данный момент .NET Framework поддерживается на большинстве современных ОС Windows (Windows 10/8.1/8/7/Vista). А благодаря проекту Mono можно создавать приложения, которые будут работать и на других ОС семейства Linux, в том числе на мобильных платформах Android и iOS.

Также еще следует отметить такую особенность языка C# и фреймворка .NET, как *автоматическая сборка мусора*. А это значит, что нам в большинстве случаев не придется, в отличие от С++, заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда CLR сама вызовет сборщик мусора и очистит память.

*JIT-компиляция*

Код на C# компилируется в приложения или сборки с расширениями .exe или .dll на языке CIL. Далее при запуске на выполнение подобного приложения происходит JIT-компиляция (Just-In-Time) в машинный код, который затем выполняется. При этом, поскольку наше приложение может быть большим и содержать много инструкций, в текущий момент времени будет компилироваться лишь та часть приложения, к которой непосредственно идет обращение. Если мы обратимся к другой части кода, то она будет скомпилирована из CIL в машинный код. При том уже скомпилированная часть приложения сохраняется до завершения работы программы. В итоге это повышает производительность.

**2.2 Описание среды разработки Visual Studio 2015**

Microsoft Visual Studio 2015 — это набор инструментов для создания программного обеспечения: от планирования до разработки пользовательского интерфейса, написания кода, тестирования, отладки, анализа качества кода и производительности. Эти инструменты предназначены для максимально эффективной совместной работы; все они доступны в интегрированной среде разработки (IDE) Visual Studio.

Visual Studio можно использовать для создания различных типов приложений, от простых приложений для магазина и игр для мобильных клиентов до больших и сложных систем, обслуживающих предприятия и центры обработки данных.

Можно создавать:

* приложения и игры, которые выполняются не только на платформе Windows, но и на Android и iOS;
* веб-сайты и веб-службы на основе ASP.NET, JQuery, AngularJS и других популярных платформ;
* приложения для самых разных платформ и устройств, включая, но не ограничиваясь: Office, Sharepoint, Hololens, Kinect;
* игры и графические приложения для разных устройств Windows, включая Xbox, с поддержкой DirectX.

По умолчанию Visual Studio обеспечивает поддержку C#, C и C++, JavaScript, F# и Visual Basic.

* 1. **Описание технологии WPF**

*Windows Presentation Foundation (WPF)* – это технология для построения клиентских приложений Windows, являющаяся частью платформы .NET. WPF разработана как альтернатива технологии Windows Forms.

*Основные особенности технологии WPF*:

1. *Собственные методы построения и рендеринга элементов*.

В Windows Forms классы для элементов управления делегируют функции отображения системным библиотекам, таким как user32.dll. В WPF любой элемент управления полностью строится (рисуется) самой WPF. Для аппаратного ускорения рендеринга применяется технология DirectX.

1. *Независимость от разрешения устройства вывода*.

Для указания размеров в WPF используется собственная единица измерения, равная 1/96 дюйма. Кроме этого, технология WPF ориентирована на использование не пиксельных, а векторных примитивов.

1. *Декларативный пользовательский интерфейс.*

В WPF визуальное содержимое окна можно полностью описать на языке XAML. Это язык разметки, основанный на XML. Так как описание пользовательского интерфейса отделено от кода, дизайнеры могут использовать профессиональные инструменты (например, Microsoft Expression Blend), чтобы редактировать файлы XAML, улучшая внешний вид приложения. Применение XAML является предпочтительным, но не обязательным – приложения WPF можно создавать, используя только код.

1. *Веб-подобная модель компоновки*.

WPF поддерживает гибкий визуальный поток, размещающий элементы управления на основе их содержимого. В результате получается пользовательский интерфейс, который может быть адаптирован для отображения динамичного содержимого.

1. *Стили и шаблоны*.

Стили стандартизируют форматирование и позволяют повторно использовать его по всему приложению. Шаблоны дают возможность изменить способ отображения любых элементов управления, даже таких основополагающих, как кнопки или поля ввода.

1. *Анимация*.

В WPF анимация – неотъемлемая часть программного каркаса. Анимация определяется декларативными дескрипторами, и WPF запускает её в действие автоматически.

1. *Приложения на основе страниц*.

В WPF можно строить приложения с кнопками навигации, которые позволяют перемещаться по коллекции страниц.

1. **Алгоритм работы программы**

Программа условно поделена на две части: деинсталлятор и управление автозагрузкой. После запуска приложения происходит поиск всех установленных программ и программ, добавленных в автозагрузку. Вся информация о программах берется из реестра. Затем полученные данные добавляются в таблицы для деинсталлятора и для управления автозагрузкой. После этого с программами можно выполнять любые из этих действий: удаление с жесткого диска, поиск остаточных файлов, добавление и удаление из автозагрузки.

Покажем, что представляет собой алгоритм поиска в коде программы. Начнем с описания классов, в объектах которых будет хранится информация о приложениях. Для представления установленной программы и программы, добавленной в автозагрузку, были созданы 2 класса: InstalledApp и AutorunApp соответственно. Эти классы являются потомками класса BaseApp. Код классов представлен ниже.

public class BaseApp

{

public string Name { get; set; }

public ImageSource DisplayIcon { get; set; }

}

public class InstalledApp : BaseApp

{

public long Size { get; }

public DateTime? InstalledDateTime { get; set; }

public string Version { get; set; }

public string Publisher { get; set; }

public string Location { get; set; }

public string UninstallString { get; set; }

public bool IsSystemComponent { get; set; }

public string RegistryKey { get; set; }

}

public class AutorunApp : BaseApp

{

public string RegistryValueName { get; set; }

public string Source { get; set; }

public string SourceType => Source.Split('\\')[0];

public string ExePath { get; set; }

public Status Status { get; set; }

}

Перейдем непосредственно к алгоритму поиска.

* Поиск установленных программ

В методе GetAllPrograms происходит поиск программ, установленных для текущего пользователя и для всех пользователей. Это выполняется путем вызова метод GetSomePrograms и передаче ему в качестве параметра ключа реестра. По этому ключу и будет производиться поиск.

public static List<InstalledApp> GetAllPrograms()

{

//all users x64

var hklm = RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Registry64);

var programs = GetSomePrograms(hklm.OpenSubKey(RegistryWork.UNINSTALL\_KEY));

//all users x32 programs.AddRange(GetSomePrograms(Registry.LocalMachine.OpenSubKey(RegistryWork.UNINSTALL\_KEY)));

//current user programs.AddRange(GetSomePrograms(Registry.CurrentUser.OpenSubKey(RegistryWork.UNINSTALL\_KEY)));

programs.Sort((x, y) => string.Compare(x.Name.ToLower(), y.Name.ToLower(), StringComparison.Ordinal));

return programs;

}

private static List<InstalledApp> GetSomePrograms(RegistryKey registryKey)

{

var programs = new List<InstalledApp>();

foreach (string skName in registryKey.GetSubKeyNames())

{

using (RegistryKey subKey = registryKey.OpenSubKey(skName))

{

string name = RegistryWork.GetDisplayName(subKey);

bool isSystemComponent = RegistryWork.IsSystemComponent(subKey);

if (name != "" && isSystemComponent == false && RegistryWork.IsHaveParent(subKey) == false)

{

ImageSource displayIcon = RegistryWork.GetDisplayIcon(subKey);

long size = RegistryWork.GetSize(subKey);

DateTime? installDate = GetInstallDate(subKey);

object version = subKey.GetValue("DisplayVersion");

object publisher = subKey.GetValue("Publisher");

string installLocation = RegistryWork.GetInstallLocation(subKey);

string uninstallString = RegistryWork.GetUninstallString(subKey);

string rKey = subKey.Name;

var program = new InstalledApp(name, displayIcon, size, installDate, version, publisher, installLocation,

uninstallString, isSystemComponent, rKey);

programs.Add(program);

}

}

}

return programs;

}

* Поиск программ, добавленных в автозапуск

public static List<AutorunApp> GetAllAutorunApps()

{

//autorun for current user

var autorunApps = GetSomeAutorunApps(Registry.CurrentUser);

//autorun for all users

var hklm = RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Registry64);

autorunApps.AddRange(GetSomeAutorunApps(hklm));

var programs = GetDisabledProgramsFromFile();

autorunApps.AddRange(programs);

autorunApps.Sort((x, y) => string.Compare(x.Name.ToLower(), y.Name.ToLower(), StringComparison.Ordinal));

return autorunApps;

}

private static List<AutorunApp> GetSomeAutorunApps(RegistryKey key)

{

var autorunApps = new List<AutorunApp>();

using (RegistryKey registryKey = key.OpenSubKey(RegistryWork.AUTO\_RUN\_KEY, false))

{

foreach (string valueName in registryKey.GetValueNames())

{

string exePath = registryKey.GetValue(valueName).ToString();

if (IsFileExists(exePath))

{

string name = RegistryWork.GetNameExe(exePath) ?? valueName;

string source = registryKey.Name;

var program = new AutorunApp(name, valueName, source, exePath);

autorunApps.Add(program);

}

}

}

return autorunApps;

}

1. **Разработка приложения**

Данный курсовой проект написан на языке программирования C# в среде разработки Visual Studio 2015.

Разработка приложения начиналась с разработки интерфейса главного окна. Добавление различных компонентов в окно происходило с использованием XAML-редактора.

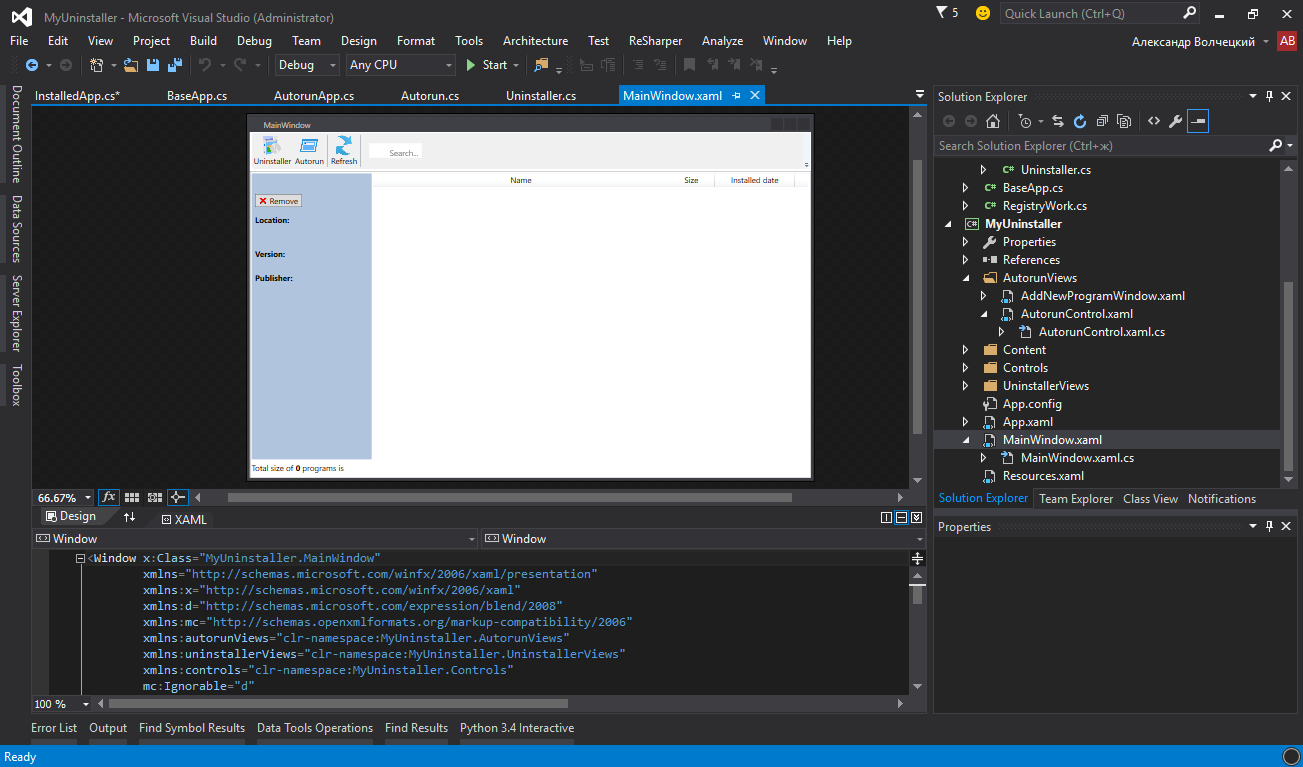


Рис. 1 – XAML-редактор Visual Studio

Как уже упоминалось выше, программа поделена на две части: деинсталлятор и управление автозагрузкой. Поэтому для каждой части был создан отдельный компонент – UserControl (пользовательский элемент управления). Затем эти 2 элемента управления были добавлены на главное окно. Такой подход упрощает разработку, так как при изменении кода одного компонента мне не приходилось думать, повлияет ли это на работу другого компонента.

1. **Описание основных функций приложения**

Продемонстрируем как выглядят в коде основные функции приложения:

* Поиск остаточных файлов в реестре

private static Dictionary<RegistryKey, string> matches;

public static Dictionary<RegistryKey, string> FindResidualFilesRegistry(InstalledApp installedApp)

{

matches = new Dictionary<RegistryKey, string>();

string key = RegistryWork.GetRegistryKeyNameWithoutPath(installedApp.RegistryKey);

List<RegistryKey> keys = new List<RegistryKey>

{

RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Registry64).OpenSubKey(RegistryWork.SOFTWARE\_KEY),

Registry.LocalMachine.OpenSubKey(RegistryWork.SOFTWARE\_KEY),

RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Registry64).OpenSubKey(RegistryWork.WOW6432\_NODE\_KEY),

Registry.LocalMachine.OpenSubKey(RegistryWork.WOW6432\_NODE\_KEY)

};

foreach (var registryKey in keys)

{

FindRegistryNotRecursive(key, registryKey);

}

List<RegistryKey> keysRecursive = new List<RegistryKey>

{

Registry.CurrentUser,

RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Registry64).OpenSubKey(RegistryWork.UNINSTALL\_KEY),

Registry.LocalMachine.OpenSubKey(RegistryWork.UNINSTALL\_KEY),

};

foreach (var registryKey in keysRecursive)

{

FindRegistryRecursive(key, registryKey);

}

return matches;

}

private static void FindRegistryNotRecursive(string keyName, RegistryKey registryKey)

{

//search by keys

var subKeysNames = registryKey.GetSubKeyNames();

foreach (var subKeysName in subKeysNames)

{

var subKey = registryKey.OpenSubKey(subKeysName);

if (RegistryWork.GetRegistryKeyNameWithoutPath(subKey.Name).ToLower().Contains(keyName.ToLower()))

{

matches.Add(subKey, null);

}

}

//search by values

if (RegistryWork.GetRegistryKeyNameWithoutPath(registryKey.Name).ToLower().Contains(keyName.ToLower()) == false)

{

var valueNames = registryKey.GetValueNames();

foreach (var valueName in valueNames)

{

if (registryKey.GetValue(valueName)?.ToString().ToLower().Contains(keyName.ToLower()) == true)

{

matches.Add(registryKey, valueName + " [" + registryKey.GetValue(valueName) + "]");

}

}

}

}

private static void FindRegistryRecursive(string keyName, RegistryKey registryKey)

{

FindRegistryNotRecursive(keyName, registryKey);

var subKeyNames = registryKey.GetSubKeyNames();

foreach (var subKeyName in subKeyNames)

{

try

{

FindRegistryRecursive(keyName, registryKey.OpenSubKey(subKeyName));

}

catch (SecurityException)

{

}

}

}

* Поиск остаточных файлов в файловой системе

public static List<DirectoryInfo> FindResidualFiles(InstalledApp installedApp)

{

var files = new List<DirectoryInfo>();

if (Directory.Exists(installedApp.Location))

{

var directory = new DirectoryInfo(installedApp.Location);

files.Add(directory);

}

return files;

}

* Удаление программы

private void RemoveButton\_OnClick(object sender, RoutedEventArgs e)

{

if (programsListView.SelectedItem != null)

{

var uninstallPath = RegistryWork.ExtractPath(((InstalledApp)programsListView.SelectedItem).UninstallString);

string fileName = uninstallPath, arguments = "";

if (uninstallPath.ToLower().Contains("msiexec.exe"))

{

var a = uninstallPath.Split(' ');

fileName = a[0];

arguments = a[1];

}

var process = new Process

{

StartInfo = new ProcessStartInfo

{

FileName = fileName, Arguments = arguments

}

};

process.Start();

IsEnabled = false;

Wait(process.ProcessName);

IsEnabled = true;

if (IsDeleted(fileName, arguments))

{

FindResidualFiles();

Refresh();

}

}

}

* Добавление в автозапуск

private void AddProgramButton\_OnClick(object sender, RoutedEventArgs e)

{

AddNewProgramWindow addNewProgramWindow = new AddNewProgramWindow();

addNewProgramWindow.ShowDialog();

Refresh();

}

private void AddButton\_Click(object sender, RoutedEventArgs e)

{

string name = RegistryWork.GetNameExe(ExePathTextBox.Text);

Autorun.AddProgramToAutorun(name, (AutorunTypeComboBox.SelectedItem as ComboBoxItem).Content.ToString(), ExePathTextBox.Text);

if (isFromContextMenu)

{

MessageBox.Show("The program was added to autorun correctly");

}

Close();

}

public static void AddProgramToAutorun(string appName, string autorunType, string exePath)

{

RegistryKey registryKey = GetRegistryKeyForAutorun(autorunType);

registryKey.SetValue(appName, exePath);

}

* Удаление из автозапуска

private void DeleteButton\_OnClick(object sender, RoutedEventArgs e)

{

var program = AutorunProgramsListView.SelectedItem as AutorunApp;

Autorun.RemoveProgramFromEverywhere(program);

Refresh();

}

public static void RemoveProgramFromEverywhere(AutorunApp app)

{

RemoveProgramFromAutorun(app);

var programs = GetDisabledProgramsFromFile();

programs = programs.Where(p => p.Name != app.Name).ToList();

SetDisabledProgramsToFile(programs);

}

public static void RemoveProgramFromAutorun(AutorunApp app)

{

RegistryKey registryKey = GetRegistryKeyForAutorun(app.SourceType);

if (registryKey.GetValue(app.RegistryValueName) != null)

{

registryKey.DeleteValue(app.RegistryValueName);

var disabledApps = GetDisabledProgramsFromFile();

app.Status = Status.Disabled;

disabledApps.Add(app);

SetDisabledProgramsToFile(disabledApps);

}

}

1. **Инструкция по использованию программного продукта**

Приложение имеет минималистичный и понятный интерфейс, который можно увидеть на представленных ниже скриншотах (Рис. 2 - 9).

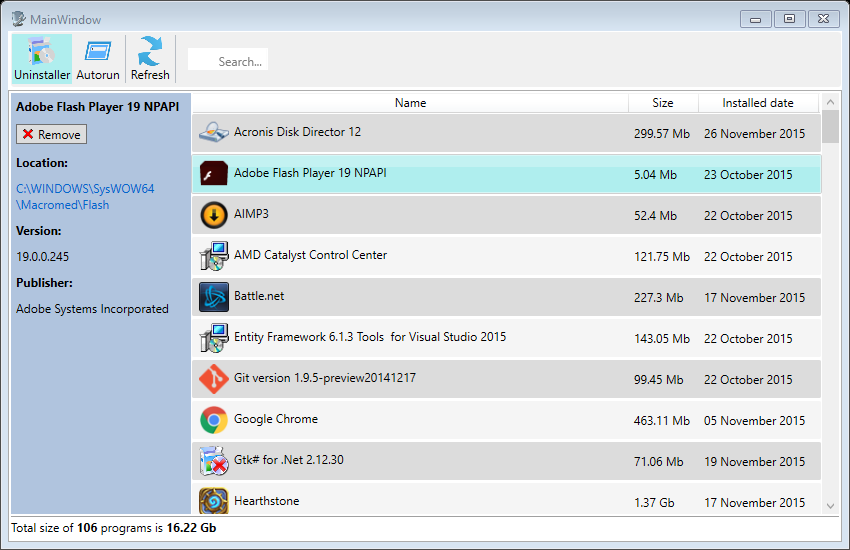
* После запуска программы пользователь попадает на панель деинсталлятора. 

Рис. 2 – Панель деинсталлятора

* Удаление программы

1) Для удаления программы нужно выделить ее в списке

2) Затем нажать кнопку “Remove”

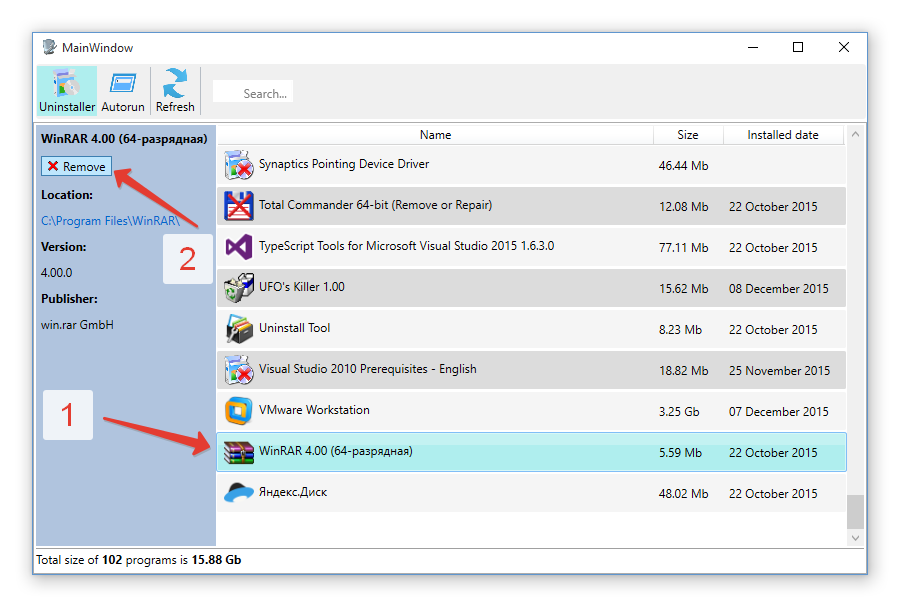


Рис. 3 – Удаление программы

* После удаления программы прозойдет поиск остаточных файлов.

А) Для удаления остаточных файлов нужно их отметить галочкой и затем нажать кнопку Remove

Б) В ином случае нужно нажать кнопку Cancel

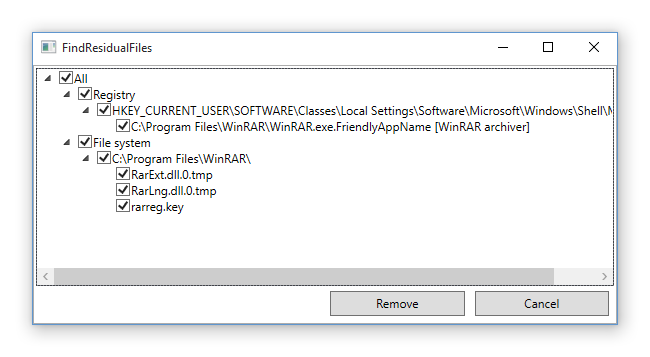


Рис. 4 – Поиск остаточных файлов

* Панель управления автозагрузкой

Для того, чтобы выбрать эту панель нужно нажать левой клавишей мыши на кнопку Autorun.

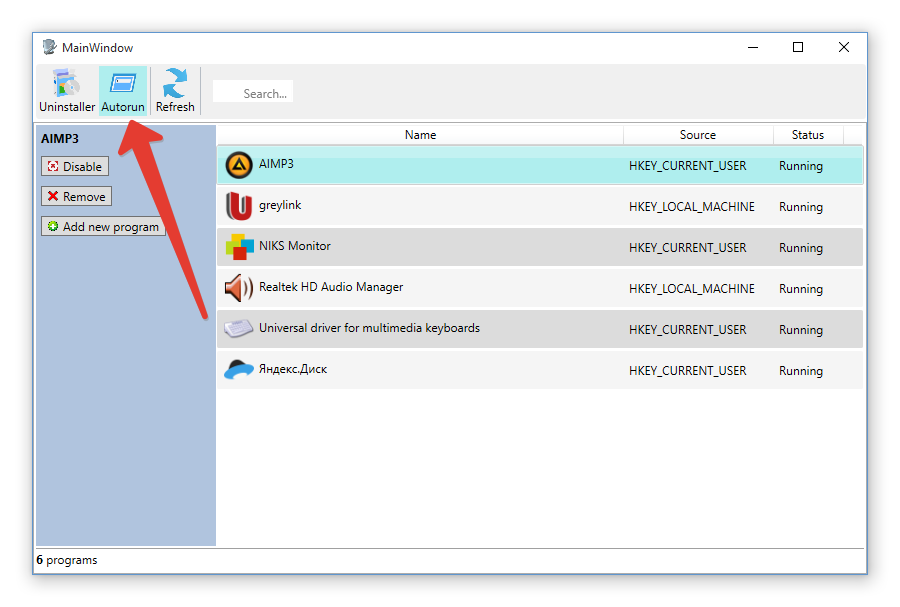


Рис. 5 – Панель управления автозагрузкой

* Отключить автозапуск

1. Выделить программу
2. Нажать кнопку Disable (программа удалиться из автозапуска, но не удалится из списка, который показан на скриншоте. Ее всегда можно будет вернуть в автозапуск нажав на эту же кнопку снова)

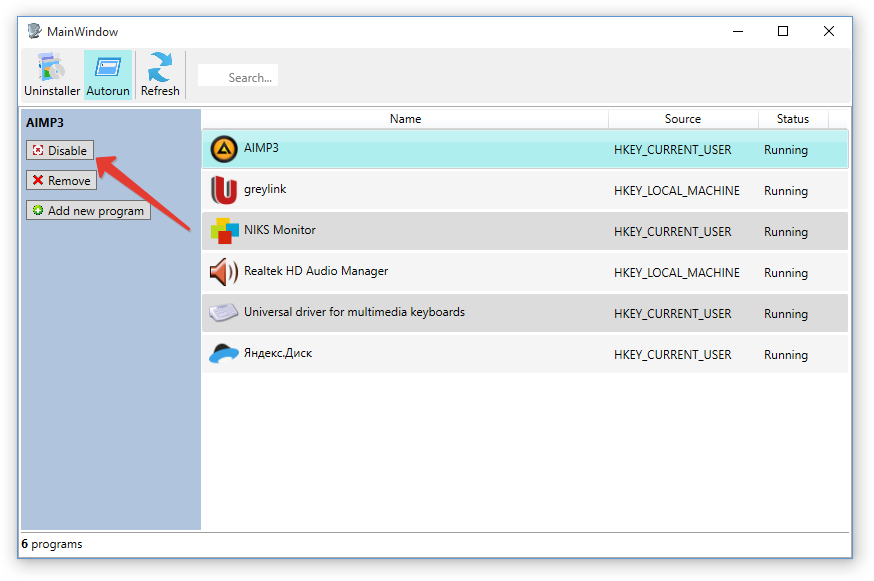
****

Рис. 6 – Отключение автозапуска

* Полное отключение автозапуска

1. Выделить программу
2. Нажать кнопку Remove (программа удалиться из автозапуска и из списка. Ее можно будет вернуть в автозапуск нажав кнопку Add new program)

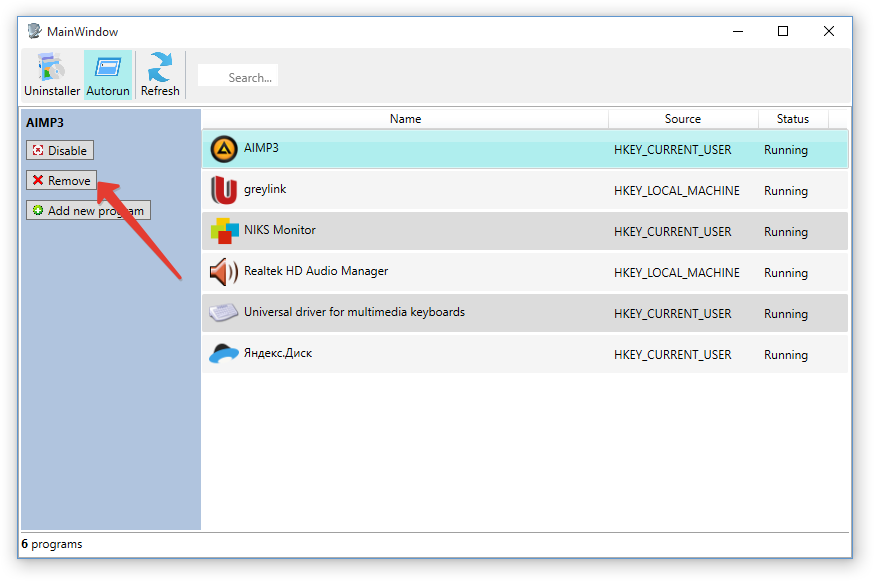


Рис. 7 – Полное отключение автозапуска программы

* Добавление программы в автозапуск

Нажать кнопку Add new program

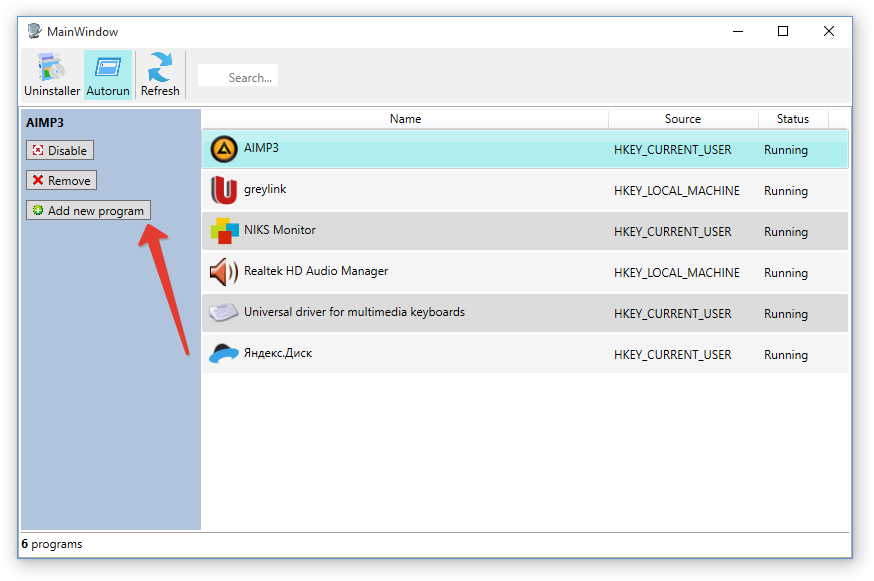


Рис. 8 – Добавление новой программы в автозапуск

Появится окно.

1. Выбрать установить для текущего или для всех пользователей.
2. Указать путь к .exe файлу
3. Нажать кнопку Add

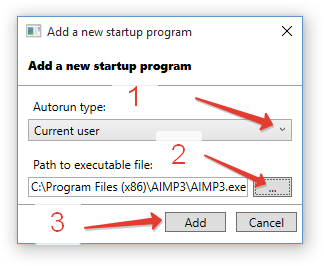


Рис. 9 – Окно добавления программы

**Заключение**

В данном курсовом проекте ставилась цель разработать деинсталлятор программ. Считаю, что данная цель достигнута. Мною было разработано приложение, которое предоставляет пользователю следующие функции:

1. Удаление программ.
2. Удаление остаточных файлов из файловой системы и реестра.
3. Добавление программ в автозапуск.
4. Удаление программ из автозапуска.

В будущем возможно добавление таких функций как:

1. Слежение за установкой программ.
2. Настройки. В них можно будет изменить внешний вид приложения, язык и т.п.

# **Список используемых источников**

1. Stackoverflow <http://stackoverflow.com/>

2. CodeProject <http://www.codeproject.com/>

3. Изучаем WPF: <http://metanit.com/sharp/wpf/>

4. WPF - Windows Presentation Foundation <http://professorweb.ru/my/WPF/base_WPF/level1/info_WPF.php>