**Инсталлятор**

**Текстпрограммы**

2017

#### АННОТАЦИЯ

В данном документе приведен текст программы «Инсталлятор», представляющей из себя программную реализацию инсталлятора устанавливающего приложения.

Исходным языком, используемым для данной разработки, является С++. Результатом компиляции и сборки исходных кодов является exe файл, готовый к исполнению.

**Оглавление**

[1. Текст программы 4](#_Toc484875895)

1. Текст программы

Листинг 1: Файл Installer.cpp

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <sstream>  #include <string>  #include <windows.h>  #include "cstringt.h"  #include "conio.h"  #include "io.h"  #include "stdlib.h"  #include "iostream"  #include "locale.h"  #include <io.h>  //////////////////////////////  #include <stdio.h>  #include <sys/stat.h>  //////////////////////////////  using namespace std;  string desktopway, Programs, Startmenu, PFDir;  string prog\_name, FirstProgWay,exefile;  /////////////////////////////////  string globalpath;  fstream fss("file.bin", std::ios::in | std::ios::out | std::ios::binary);  /////////////////////////////////  void archivate(int iter, string path){  cout << "Start Archivating ... " << endl << endl;  WIN32\_FIND\_DATA FindFileData;  HANDLE hf;  iter++;  string finalpath = globalpath + path + "\\\*";  LPSTR s = const\_cast<char \*>(finalpath.c\_str());  hf = FindFirstFile(s, &FindFileData);  if (hf != INVALID\_HANDLE\_VALUE){  do{  for (int i = 0; i < iter; i++)  printf(" ");  if (FindFileData.dwFileAttributes&FILE\_ATTRIBUTE\_DIRECTORY){  string filenamedir = FindFileData.cFileName;  if (filenamedir != ""){  if (filenamedir != "."){  if (filenamedir != ".."){  printf("%s\n", FindFileData.cFileName);  string path2 = path + "\\" + FindFileData.cFileName;  string path22 = path + "\\" + FindFileData.cFileName;  //cout << path2 << endl;  fss.write("#", 1);  fss.write("d", 1);  fss.write("#", 1);  char \*cstr = new char[path2.length() + 1];  strcpy(cstr, path2.c\_str());  for (int i = 0; i < path2.length(); i++){  //cout << byte;  fss.write(&cstr[i], sizeof(char));  }  fss.write("#", 1);  fss.write("\r", 1);  fss.write("\n", 1);  archivate(iter, path22);  }  }  }  }  else{  string statpath, statpathwr;  if (iter == 0){  statpath = globalpath + path + "\\" + FindFileData.cFileName;  statpathwr = globalpath + path + "\\\\" + FindFileData.cFileName;  }  else{  statpath = globalpath + "\\" + path + "\\" + FindFileData.cFileName;  statpathwr = globalpath + "\\\\" + path + "\\\\" + FindFileData.cFileName;  }  struct stat results;  stat(statpath.c\_str(), &results);  //cout << results.st\_size ;  fss.write("#", 1);  fss.write("f", 1);  fss.write("#", 1);  string path2 = path + "\\" + FindFileData.cFileName;  char \*cstr = new char[path2.length() + 1];  strcpy(cstr, path2.c\_str());  for (int i = 0; i < path2.length(); i++){  //cout << byte;  fss.write(&cstr[i], sizeof(char));  }  fss.write("#", 1);  int a = results.st\_size;  string sizeInt = to\_string(a);  char \*cstrInt = new char[sizeInt.length() + 1];  strcpy(cstrInt, sizeInt.c\_str());  for (int i = 0; i <sizeInt.length(); i++){  //cout << byte;  fss.write(&cstrInt[i], sizeof(char));  }  fss.write("#", 1);  fss.write("\r", 1);  fss.write("\n", 1);  fstream fs(statpath.c\_str(), std::ios::in | std::ios::binary);  if (fs.is\_open())  {  char \*byte = new char[1];  for (int i = 0; i < results.st\_size; i++){  fs.read(&byte[0], sizeof(char));  //cout << byte;  fss.write(&byte[0], 1);  }  fs.close();  }  printf(" %s\n", FindFileData.cFileName);  }  } while (FindNextFile(hf, &FindFileData) != 0);  FindClose(hf);  }  cout << "Archivating ended sucsess !" << endl << endl;  }  //////////////////////////////////////////  void FreeSpace()  {  DWORD FreeBytesAvailable;  DWORD TotalNumberOfBytes;  DWORD TotalNumberOfFreeBytes;  BOOL GetDiskFreeSpaceFlag = GetDiskFreeSpaceEx(  "c:\\", // directory name  (PULARGE\_INTEGER)&FreeBytesAvailable, // bytes available to caller  (PULARGE\_INTEGER)&TotalNumberOfBytes, // bytes on disk  (PULARGE\_INTEGER)&TotalNumberOfFreeBytes // free bytes on disk  );  if (GetDiskFreeSpaceFlag != 0)  {  cout << " Total Number Of Free Bytes = " <<(unsigned long)TotalNumberOfFreeBytes  << "( " << double(unsigned long(TotalNumberOfFreeBytes)) / 1024 / 1000  << " Mb )" << endl;  cout << " Total Number Of Bytes = " <<(unsigned long)TotalNumberOfBytes  << "( " << double(unsigned long(TotalNumberOfBytes)) / 1024 / 1000  << " Mb )" << endl;  }  else cout << " Not Present (GetDiskFreeSpace)" << endl;  }  void vivod(\_finddata\_t vfdata)  {  cout << "file\_name :" << vfdata.name << "\n";  exefile=vfdata.name;  }  void Findexe()  {  char \*fpath = "I:\\1\\\*.exe";  long ctrl, next;  int i = 1;  \_finddata\_t fdata;  ctrl = \_findfirst(fpath, &fdata);  //cout << i << ": " << "ctrl= " << ctrl << "\n";  vivod(fdata);  while (next != -1)  {  next = \_findnext(ctrl, &fdata);  if (next != -1) {  //cout << ++i << ": " << "ctrl = " << ctrl;  //cout << " next = " << next << "\n";  vivod(fdata);  };  }  \_findclose(ctrl);  \_findclose(next);  //system("pause");  }  void Copy(LPCTSTR szInDirName, LPCTSTR szOutDirName, bool flag = false)  {  WIN32\_FIND\_DATA ffd;  HANDLE hFind;  TCHAR szFind[MAX\_PATH + 1];  TCHAR szInFileName[MAX\_PATH + 1];  TCHAR szOutFileName[MAX\_PATH + 1];  lstrcpy(szFind, szInDirName);  lstrcat(szFind, "\\\*.\*"); //ищем файлы с любым именем и рысширением  hFind = FindFirstFile(szFind, &ffd);  do  {  //Формируем полный путь (источник)  lstrcpy(szInFileName, szInDirName);  lstrcat(szInFileName, "\\");  lstrcat(szInFileName, ffd.cFileName);  //Формируем полный путь (результат)  lstrcpy(szOutFileName, szOutDirName);  lstrcat(szOutFileName, "\\");  lstrcat(szOutFileName, ffd.cFileName);  if (flag) //если flag == true, то копируем и папки  {  if (ffd.dwFileAttributes & 0x00000010)  {  if (lstrcmp(ffd.cFileName, ".") == 0 ||  lstrcmp(ffd.cFileName, "..") == 0) continue;  CreateDirectory(szOutFileName, NULL);  Copy(szInFileName, szOutFileName);  }  } //иначе пропускаем папки  else  if (ffd.dwFileAttributes & 0x00000010) continue;  CopyFile(szInFileName, szOutFileName, TRUE);  } while (FindNextFile(hFind, &ffd));  FindClose(hFind);  }  int ReadInfo()  {  cout << "Reading information from the registry ... " << endl << endl;  CHAR szPath0[] = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\";  CHAR szBuf0[MAX\_PATH];  DWORD dwBufLen0 = MAX\_PATH;  if (RegGetValue(HKEY\_LOCAL\_MACHINE, szPath0, "ProgramFilesDir", RRF\_RT\_REG\_SZ, NULL, (BYTE\*)szBuf0, &dwBufLen0) != ERROR\_SUCCESS){  cout << "Error When reading a string" << endl;    }  cout << "The folder ProgramFiles located :" << szBuf0 << endl;  PFDir = szBuf0;    CHAR szPath[] = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Shell Folders\\";    CHAR szBuf1[MAX\_PATH];  DWORD dwBufLen = MAX\_PATH;  if (RegGetValue(HKEY\_CURRENT\_USER, szPath, "Desktop", RRF\_RT\_REG\_SZ, NULL, (BYTE\*)szBuf1, &dwBufLen) != ERROR\_SUCCESS){  cout << "Error When reading a string" << endl;  //return 4;  }  cout << "The folder Desktop located :" <<szBuf1 << endl;  desktopway = szBuf1;    /\*LPCTSTR link1 = szBuf1;  cout << "Desktop is link1 " << link1 << endl;\*/  CHAR szBuf2[MAX\_PATH];  DWORD dwBufLen2 = MAX\_PATH;  if (RegGetValue(HKEY\_CURRENT\_USER, szPath, "Programs", RRF\_RT\_REG\_SZ, NULL, (BYTE\*)szBuf2, &dwBufLen2) != ERROR\_SUCCESS){  cout << "Error When reading a string" << endl;  //return 4;  }  cout << "The folder Programs located :" << szBuf2 << endl;  Programs = szBuf2;    CHAR szBuf3[MAX\_PATH];  DWORD dwBufLen3 = MAX\_PATH;  if (RegGetValue(HKEY\_CURRENT\_USER, szPath, "Start Menu", RRF\_RT\_REG\_SZ, NULL, (BYTE\*)szBuf3, &dwBufLen3) != ERROR\_SUCCESS){  cout << "Error When reading a string" << endl;    }  cout << "The folder Start Menu located :" << szBuf3 << endl;  Startmenu=szBuf3 ;  cout << "Registry reading completed successfully !" << endl << endl;  return 4;  }  int Uninst()  {  cout << "Adding a program to the Uninstall list ..." << endl;    std::string way, way2, strUninstName, strUninstName2;  // Строка которую будем писать в реестр  //UninstallString    \_TCHAR Publisher[] = "HELLEN";  \_TCHAR \*szTestString = new \_TCHAR[prog\_name.size() + 1];  szTestString[prog\_name.size()] = 0;  std::copy(prog\_name.begin(), prog\_name.end(), szTestString);    way = "SOFTWARE\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\Uninstall\\";  way2 = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall\\";  strUninstName = way + prog\_name;  strUninstName2 = way2 + prog\_name;  LPCTSTR lp\_UninstName = strUninstName.c\_str();  HKEY HKeyUninstall;  RegCreateKey(HKEY\_LOCAL\_MACHINE, lp\_UninstName, &HKeyUninstall);//SEE WOW6432NODE      // Пишем тестовую строку в созданный ключ  if (RegSetValueEx(HKeyUninstall, \_T("DisplayName"), 0, REG\_SZ, (BYTE\*)szTestString, 30) != ERROR\_SUCCESS){  cout << \_T("Error writing to the key ") << endl;  return 2;  }  if (RegSetValueEx(HKeyUninstall, \_T("UninstallString"), 0, REG\_SZ, (BYTE\*)szTestString, 30) != ERROR\_SUCCESS){  cout << \_T("Error writing to the key ") << endl;  return 2;  }  if (RegSetValueEx(HKeyUninstall, \_T("Publisher"), 0, REG\_SZ, (BYTE\*)Publisher, 30) != ERROR\_SUCCESS){  cout << \_T("Error writing to the key ") << endl;  return 5;  }  // Закрываем описатель ключа  if (RegCloseKey(HKeyUninstall) != ERROR\_SUCCESS){  cout << \_T("Error when closing the key ") << endl;  return 3;  };  LPCTSTR lp\_UninstName2 = strUninstName2.c\_str();  HKEY HKeyUninstall2;  //RegCreateKey(HKEY\_CURRENT\_USER, lp\_UninstName2, &HKeyUninstall2);  //// Пишем тестовую строку в созданный ключ  //if (RegSetValueEx(HKeyUninstall2, \_T("DisplayName"), 0, REG\_SZ, (BYTE\*)szTestString, 30) != ERROR\_SUCCESS){  // cout << \_T("Error writing to the key ") << endl;  // return 5;  //}  //if (RegSetValueEx(HKeyUninstall2, \_T("UninstallString"), 0, REG\_SZ, (BYTE\*)szTestString, 30) != ERROR\_SUCCESS){  // cout << \_T("Error writing to the key ") << endl;  // return 5;  //}  //// Закрываем описатель ключа  //if (RegCloseKey(HKeyUninstall2) != ERROR\_SUCCESS){  // cout << \_T("Error when closing the key ") << endl;  // return 9;  //};  RegCreateKey(HKEY\_LOCAL\_MACHINE, lp\_UninstName2, &HKeyUninstall2);//SEE WOW6432NODE  // Пишем тестовую строку в созданный ключ  if (RegSetValueEx(HKeyUninstall2, \_T("DisplayName"), 0, REG\_SZ, (BYTE\*)szTestString, 30) != ERROR\_SUCCESS){  cout << \_T("Error writing to the key ") << endl;  return 5;  }  if (RegSetValueEx(HKeyUninstall2, \_T("UninstallString"), 0, REG\_SZ, (BYTE\*)szTestString, 30) != ERROR\_SUCCESS){  cout << \_T("Error writing to the key ") << endl;  return 5;  }  if (RegSetValueEx(HKeyUninstall2, \_T("Publisher"), 0, REG\_SZ, (BYTE\*)Publisher, 30) != ERROR\_SUCCESS){  cout << \_T("Error writing to the key ") << endl;  return 5;  }  // Закрываем описатель ключа  if (RegCloseKey(HKeyUninstall2) != ERROR\_SUCCESS){  cout << \_T("Error when closing the key ") << endl;  return 9;  };  cout << "Uninstall list - Completed successfully !" << endl;  return 15;  }  int link()  {  cout << "Create a Icon on the desktop ..." << endl << endl;    string linkname = desktopway + "\\" + exefile;  LPCTSTR lp\_linkname = linkname.c\_str();  string endlink = PFDir + "\\"+prog\_name+"\\" + exefile;  LPCTSTR lp\_endlink = endlink.c\_str();    CreateSymbolicLink(lp\_linkname, lp\_endlink, 0);    string Progr = Programs + "\\" + exefile;  LPCTSTR lp\_Programs = Progr.c\_str();  string Stmen = Startmenu + "\\" + exefile;  LPCTSTR lp\_Startmenu = Stmen.c\_str();  cout << "Create a Icons on the start and Programs menu..." << endl << endl;  CreateSymbolicLink(lp\_Programs, lp\_endlink, 0);  CreateSymbolicLink(lp\_Startmenu, lp\_endlink, 0);  cout << "Icons created." << endl << endl;  return 25;  }  int reg()  {  cout << "Recording information in the registry ..." << endl;  std::string s1, s3;  s1 = "SOFTWARE\\";  s3 = s1 + prog\_name;  LPCTSTR lp\_s3 = s3.c\_str();  HKEY HKey1;  RegCreateKey(HKEY\_CURRENT\_USER, lp\_s3, &HKey1);  cout << "HKEY\_CURRENT\_USER - Completed successfully ! " << endl;  RegCreateKey(HKEY\_LOCAL\_MACHINE, lp\_s3, &HKey1);//SEE WOW6432NODE  cout << "HKEY\_LOCAL\_MACHINE - Completed successfully !" << endl;  return 10;  }  void unzip(string s2){  globalpath = s2;  char byte;  string byteS;  string sizeFile = "";  string nameFile = "";  while (fss.read(&byte, sizeof(char))){  //#f#\CsgoM.vcxproj#4211#  //#d#\Debug#  byteS = string(1, byte);  if (byteS == "#"){  fss.read(&byte, sizeof(char));  byteS = string(1, byte);  if (byteS == "d"){  fss.read(&byte, sizeof(char));  fss.read(&byte, sizeof(char));  byteS = string(1, byte);  while (byteS != "#"){  nameFile.push\_back(byte);  fss.read(&byte, sizeof(char));  byteS = string(1, byte);    }  string filePath = globalpath + nameFile;  CreateDirectory(filePath.c\_str(), NULL);  fss.read(&byte, sizeof(char));  fss.read(&byte, sizeof(char));  cout << nameFile << endl;  cout << filePath << endl;  nameFile = "";  }  if (byteS == "f"){  fss.read(&byte, sizeof(char));  fss.read(&byte, sizeof(char));  byteS = string(1, byte);  while (byteS != "#"){  nameFile.push\_back(byte);    fss.read(&byte, sizeof(char));  byteS = string(1, byte);  }  fss.read(&byte, sizeof(char));  byteS = string(1, byte);  while (byteS != "#"){  sizeFile.push\_back(byte);  fss.read(&byte, sizeof(char));  byteS = string(1, byte);  }  fss.read(&byte, sizeof(char));  fss.read(&byte, sizeof(char));  string filePath = globalpath + nameFile;  ofstream ofs(filePath);  fstream fs1(filePath.c\_str(), std::ios::out | std::ios::binary);  int sizef = atoi(sizeFile.c\_str());  cout << sizef << endl;  char \*byte2 = new char[1];  for (int i = 0; i < sizef; i++){  //cout << byte;  fss.read(&byte2[0], sizeof(char));  fs1.write(&byte2[0], 1);  }  cout << nameFile << endl;  cout << filePath << endl;  nameFile = "";  sizeFile = "";  }  }  }  }  int main(int argc, char \*argv[])  {  if (argc > 2 {  string s1 = string(argv[1]);  string s2 = string(argv[2]);  WIN32\_FIND\_DATA FindFileData;  HANDLE hf;  setlocale(LC\_ALL, "Russian");  if (s1 == "-a"){  cout << "---Welcome to the Archivator !---" << endl;    ofstream ofs("file.bin");    globalpath = s2;  string path = "";  int iter = -1;  archivate(iter, path);    }  }  else  {  cout << "---Welcome to the installer---" << endl;  ReadInfo();  cout << "Enter the name of your program" << endl;  cin >> prog\_name;  //cout << "Enter the path to your program " << endl;  // cin >> FirstProgWay;  LPCTSTR lp\_s1 = FirstProgWay.c\_str();  string ProgDestWay = PFDir + "\\" + prog\_name;  LPCTSTR lp\_ProgDestWay = ProgDestWay.c\_str();  //Тут кладем программу в папку прогфайлс, а lp\_ProgDestWay содержит путь до(програм файлс)  CreateDirectory(lp\_ProgDestWay, NULL);  //Copy(lp\_s1, lp\_ProgDestWay, true);  ////////////////////////////////////////////////////////////////////////////  unzip(ProgDestWay);    ////////////////////////////////////////////////////////////////////////////  FreeSpace();  Findexe();  reg();    link();  Uninst();  cout << "Installation completed !!! " << endl;  system("pause");  }  } |