Как сохранить дизассемблерный листинг в IdaPro v.6.0 demo?

by **Erfaren**
http://erfaren.narod.ru
erfaren@rambler.ru

Введение

Ильфак Гильфанов, создатель знаменитого дизассемблера **IdaPro** не перестает удивлять нас своим творением. На этот раз мы можем тестировать его новую версию **6.0**. Она весьма существенно отличается от предыдущих переходом с платформы разработки **Borland C++** на платформу **Qt**. Внешне это выражается в более изящном и удобном интерфейсе, одинаковом для различных операционных систем. Что конечно немаловажно, но для нас более существенным является качество дизассемблирования программ. Посмотрим, что на этот раз приготовил нам наш творец. К сожалению, первое, что бросается в глаза, относится к плохой новости. **Ильфак**, в шестой версии, заблокировал нашу лазейку по сохранению листинга кода через буфер обмена. То, что он сделал это только в последней версии, заставляет подозревать его в чтении наших предыдущих статей, где мы описывали этот способ работы с дизассемблерным кодом 🙂. Т.е. копировать можно, но не более нескольких килобайт, что для многомегабайтных «простынь» кода совершенно не приемлемо. И что нам теперь делать? Как тестировать его любимое детище? Не покупать же данную программу только ради тестирования! Конечно, выход в данном случае мы все-таки найдем, обратив внимание на средства автоматизации **IdaPro**. А если наш любимый автор вырубит поддержку скриптов и плагинов в следующей версии? Допустим, мы опять найдем выход. Тогда **Ильфак** может пойти по пути фирмы **«1С»**, которая уже давно не выпускает демо-версии своих продуктов, только демо-ролики 🙂. Так что запасайтесь на всякий случай нынешними шестыми демо-версиями впрок, а то чем черт не шутит 🙂.

Средства автоматизации IdaPro

Как вы поняли, речь будет идти о скриптах и плагинах в интерактивном дизассемблере. В коммерческих версиях программно получить ассемблерный листинг **(*.asm** файл) можно с помощью строки встроенного си-подобного кода

```
WriteTxt(pFile, 0, BADADDR);  // Create the assembler file
```

либо, что то же самое,

```
GenerateFile(OFILE_ASM, pFile, 0, BADADDR, 0);
```

Для получения полного листинга (***.lst** файл) выполняем команду

```
GenerateFile(OFILE_LST, pFile, 0, BADADDR, 0);
```

Однако было бы странно, если бы этот простой код работал бы в демо-версии. И действительно, **Ильфак** не «забыл» его заблокировать. Хотя оставил генерацию ***.map** и ***.idc** файлов с помощью функций

```
WriteMap(pFile);
```

которая является макросом для

```
GenerateFile(OFILE_MAP, pFile, 0, BADADDR,  GENFLG_MAPSEGS|GENFLG_MAPNAME);
```

и

```
GenerateFile(OFILE_IDC, pFile, 0, BADADDR, 0);
```

где

```
pFile = fopen(cFileName, "w");
```

Эти и другие определения можно увидеть в файле **idc\idc.idc IdaPro**.

Конечно, без листинга кода делать нам почти нечего. Поэтому займемся поиском обходных путей его получения, рискуя, правда, оказать «услугу» **Ильфаку** по блокированию этих возможностей в следующих версиях его детища.

Работа со скриптами IdaPro

Мы не будем объяснять подробно работу скриптового движка **IdaPro**. По этой теме информации достаточно. Можно почитать книги **Криса Касперски**, **«Образ мышления – дизассемблер IdaPro»** и др. Либо **«The IDA Pro Book»,** by **Chris Eagle** и т.п. Покажем, как с помощью скриптов можно сохранить большую часть нужного нам кода.

Очень хорошей функцией является

```
string GetCurrentLine();  // Get the disassembly line at the cursor
```

Она позволяет получить строку кода из линии, в которой находится экранный курсор. Все было бы замечательно, если бы существовали средства программного перемещения курсора по экрану, либо эмуляция нажатия клавиши «стрелка вниз» в скриптовом движке **IdaPro**. На самом деле курсор мы можем двигать программно, но только от одного «узлового» адреса к другому. Здесь следует сказать, что листинг «Иды» это не последовательный набор строк, как может показаться, а «плоское» дерево. Т.е. одному «узловому» адресу может соответствовать несколько реальных строк кода и всевозможных комментариев и ссылок. Каждая строка кода одного узла может иметь различный тип, обрабатываемый индивидуально или содержать массив строк одного типа. Фактически нам предстоит разобраться с каждым «узлом» листинга, обработав независимо все его компоненты. Те компоненты узла, которые мы сможем прочесть, можно будет сохранить в файл, попутно выполнив их индивидуальную обработку, если необходимо. Но сначала разберемся, что представляет из себя «узел» строк одного виртуального адреса или **элемента** в терминологии **IdaPro**. Если мы распечатаем все линии узла, то определить следующий адрес можно с помощью команд

```
ea = NextAddr(ea);
ea = NextHead(ea, BADADDR);
ea = NextNotTail(ea);
```

а перейти к нему посредством

```
Jump(ea);
```

Функция **NextAddr** эквивалентна существующему выражению **ea++** либо **BADADDR** в противном случае. Поскольку линий текста с одним и тем же адресом **ea** может быть несколько, то функция **Jump(ea)** может перейти только на одну из них и применить программно **GetCurrentLine** мы можем только к этой «предпочтительной» строке листинга. Если адрес **ea** не существует среди отображаемых адресов в листинге, то переход произойдет на ближайший существующий адрес, не превышающий указанный. Поэтому функция **NextAddr** довольна бесполезна для данного случая. Более интересна функция **NextHead**, которая определяет адрес следующей «головы» дизассемблерного листинга. Чтобы было понятней, скажем, что **IdaPro** расщепляет линейную последовательность анализируемых байтов на «отрезки». Если этот «отрезок» есть машинная инструкция кода, то первый ее байт называется «головой», а остальные «хвостом». Кроме инструкций кода могут быть инструкции данных или даже неопределенные (нераспознанные) «отрезки» машинных байт. В этих инструкциях нет «головы» в модели **Ильфака Гильфанова**, поэтому используя функцию **NextHead**, мы рискуем пропустить узлы данных и неопределенные байты. Поэтому, остается только одна возможность, применить функцию **NextNotTail** для определения следующего отображаемого адреса узла в дизассемблерном листинге.

Теперь у нас остается очень нетривиальная задача для скриптового движка **IdaPro** получить все строки данного узла, отображаемых в листинге.

Попробуем решить эту задачу с помощью скриптов, хотя бы частично и затем проделаем ту же работу для плагинов.

Разбор узла дизассемблерного листинга IdaPro с помощью скриптов

Удобно начать работу с изучения некоторого подходящего скрипта – прототипа. Таковым мы выбрали скрипт **dumpinfo.idc** (http://www.hex-rays.com/idapro/freefiles/dumpinfo.idc). Там делается некоторая подобная работа, правда, не вся возможная информация «вытаскивается» из недр «Иды» .

Важный момент, который мы видим, это наличие в каждом узле определенного набора **32**-битных флагов, обработка которых позволяет, в принципе, получить соответствующую этим флагам информацию. Сами эти флаги находятся с помощью функции

```
flags = GetFlags(ea);
```

Использование этих флагов в принципе позволяет извлечь всю сопутствующую им информацию, однако практически осуществить это достаточно затруднительно, в силу большого разнообразия подходящих функций. Поэтому мы ограничимся шаблоном **idc**-программы, полезного для дальнейшего исследования в этом направлении, тем более, что средствами плагинов данная задача решается значительно проще.

```
//////////////////////////////////////////////////////////////////////////
// template.idc
//////////////////////////////////////////////////////////////////////////
#include <idc.idc>

static main() {
  auto ea, fn, fp, flags, sline, i, j, k, l;

  // Output To File
  fn = AskFile(1, "*.lst", "Output File Name?");  // Get *.lst File Name
  fp = fopen(fn, "w");  // Open File for Output

  if(!fp) {
    Warning("Error opening output file!\n");  // Error Opening File
    fclose(fp);
    return;
  }

  fprintf(fp, "================================================================================\n");

  i = 0;

  for(ea = MinEA(); ea != BADADDR && i++ < 100000; ea = NextNotTail(ea)) {
    Jump(ea);

    flags = GetFlags(ea);

    fprintf(fp, "%s\n", GetCurrentLine());  // Get disassembly line

    fprintf(fp, "--------------------------------------------------------------------------------\n");

    l = 0;

    if(flags & FF_LINE) {
      for (k = 998; k >= 0; k--) if (LineA(ea, k) != "") break;  // Find Last Line  // LineA() bug > 1000t

      if((sline =  LineA(ea, 0)) != "") {
        fprintf(fp, "%s\n", "FF_LINE - LineA:");
        fprintf(fp, "\t%s\n", sline);
```

```
      l = 1;
    }

    for(j = 1; j <= k; j++) fprintf(fp, "\t%s\n", LineA(ea, j));  // Get All Lines
  }

  if(l == 1) fprintf(fp, "--------------------------------------------------------------------------------\n");

  l = 0;

  if(flags & FF_LINE) {
    for (k = 998; k >= 0; k--) if (LineB(ea,k) != "") break;  // Find Last Line

    if((sline =  LineB(ea, 0)) != "") {
      fprintf(fp, "%s\n", "FF_LINE - LineB:");
      fprintf(fp, "\t%s\n", sline);
    }

    for(j = 1; j <= k; j++) fprintf(fp, "\t%s\n", LineB(ea, j));  // Get All Lines
  }

  if(l == 1) fprintf(fp, "--------------------------------------------------------------------------------\n");

  if(flags & FF_CODE) fprintf(fp, "FF_CODE - GetDisasm(ea) : %s\n", GetDisasm(ea));
  if(flags & FF_DATA) fprintf(fp, "FF_DATA - GetDisasm(ea) : %s\n", GetDisasm(ea));

  if(flags & FF_COMM) {
    if((sline = Comment(ea)) != "") fprintf(fp, "FF_COMM - Comment(ea) : %s\n", sline);
    if((sline = RptCmt(ea)) != "") fprintf(fp, "FF_COMM - RptCmt(ea) : %s\n", sline);
  }

  if(flags & FF_NAME) {
    if((sline = Name(ea)) != "") fprintf(fp, "FF_NAME - Name(ea) : %s\n", Name(ea));  // User-defined has name
    if((sline = NameEx(BADADDR, ea)) != "") fprintf(fp, "FF_NAME - NameEx(BADADDR, ea) : %s\n", sline);  // Get visible name of the byte
    if((sline = GetTrueNameEx(BADADDR, ea)) != "") fprintf(fp, "FF_NAME - GetTrueNameEx(BADADDR, ea) : %s\n", GetTrueNameEx(BADADDR,
ea));  // Get true name of the byte
  }

  if(flags & FF_FUNC) fprintf(fp, "FF_FUNC - GetFunctionName(ea): %s\n", GetFunctionName(ea));  // Function start

  if(flags & FF_LABL) fprintf(fp, "FF_LABL : Dummy has name\n");
  if(flags & FF_ANYNAME) fprintf(fp, "FF_ANYNAME = FF_LABL|FF_NAME)\n");
  if(flags & FF_IVL) fprintf(fp, "FF_IVL : Byte has value\n");
  if(flags & FF_TAIL) fprintf(fp, "FF_TAIL : Tail\n");
  if(flags & FF_UNK) fprintf(fp, "FF_UNK : Unknown\n");
  if(flags & FF_REF) fprintf(fp, "FF_REF : References\n");
  if(flags & FF_FLOW) fprintf(fp, "FF_FLOW : Exec flow from prev instruction\n");
  if(flags & FF_VAR) fprintf(fp, "FF_VAR : Byte is variable\n");
  if(flags & FF_0VOID) fprintf(fp, "FF_0VOID :Void (unknown) 1\n");
  if(flags & FF_0NUMH) fprintf(fp, "FF_0NUMH : Hexadecimal number 1\n");
  if(flags & FF_0NUMD) fprintf(fp, "FF_0NUMD : Decimal number 1\n");
  if(flags & FF_0CHAR) fprintf(fp, "FF_0CHAR : Char ('x') 1\n");
  if(flags & FF_0SEG) fprintf(fp, "FF_0SEG : Segment 1\n");
  if(flags & FF_0OFF) fprintf(fp, "FF_0OFF : Offset 1\n");
  if(flags & FF_0NUMB) fprintf(fp, "FF_0NUMB : Binary number 1\n");
  if(flags & FF_0NUMO) fprintf(fp, "FF_0NUMO : Octal number 1\n");
  if(flags & FF_0ENUM) fprintf(fp, "FF_0ENUM : Enumeration 1\n");
  if(flags & FF_0FOP) fprintf(fp, "FF_0FOP : Forced operand 1\n");
  if(flags & FF_0STRO) fprintf(fp, "FF_0STRO : Struct offset 1\n");
  if(flags & FF_0STK) fprintf(fp, "FF_0STK : Stack variable 1\n");
  if(flags & FF_0FLT) fprintf(fp, "FF_0FLT : Floating point number 1\n");
  if(flags & FF_0CUST) fprintf(fp, "FF_0CUST : Custom format type 1\n");
  if(flags & FF_1VOID) fprintf(fp, "FF_1VOID : Void (unknown) 2\n");
  if(flags & FF_1NUMH) fprintf(fp, "FF_1NUMH : Hexadecimal number 2\n");
  if(flags & FF_1NUMD) fprintf(fp, "FF_1NUMD : Decimal number 2\n");
  if(flags & FF_1CHAR) fprintf(fp, "FF_1CHAR : Char ('x') 2\n");
  if(flags & FF_1SEG) fprintf(fp, "FF_1SEG : Segment 2\n");
  if(flags & FF_1OFF) fprintf(fp, "FF_1OFF : Offset 2\n");
```

```
    if(flags & FF_1NUMB) fprintf(fp, "FF_1NUMB : Binary number 2\n");
    if(flags & FF_1NUMO) fprintf(fp, "FF_1NUMO : Octal number 2\n");
    if(flags & FF_1ENUM) fprintf(fp, "FF_1ENUM : Enumeration 2\n");
    if(flags & FF_1FOP) fprintf(fp, "FF_1FOP : Forced operand 2v");
    if(flags & FF_1STRO) fprintf(fp, "FF_1STRO : Struct offset 2\n");
    if(flags & FF_1STK) fprintf(fp, "FF_1STK : Stack variable 2\n");
    if(flags & FF_1FLT) fprintf(fp, "FF_1FLT : Floating point number 2\n");
    if(flags & FF_1CUST) fprintf(fp, "FF_1CUST : Custom format type 2\n");
    if(flags & FF_BYTE) fprintf(fp, "FF_BYTE : Byte\n");
    if(flags & FF_WORD) fprintf(fp, "FF_WORD : Word\n");
    if(flags & FF_DWRD) fprintf(fp, "FF_DWRD : Dword\n");
    if(flags & FF_QWRD) fprintf(fp, "FF_QWRD : Qword\n");
    if(flags & FF_TBYT) fprintf(fp, "FF_TBYT : Tbyte\n");
    if(flags & FF_ASCI) fprintf(fp, "FF_ASCI : ASCII\n");
    if(flags & FF_STRU) fprintf(fp, "FF_STRU : Struct\n");
    if(flags & FF_OWRD) fprintf(fp, "FF_OWRD : Octaword (16 bytes)\n");
    if(flags & FF_FLOAT) fprintf(fp, "FF_FLOAT : Float\n");
    if(flags & FF_DOUBLE) fprintf(fp, "FF_DOUBLE : Double\n");
    if(flags & FF_PACKREAL) fprintf(fp, "FF_PACKREAL : Packed decimal real\n");
    if(flags & FF_ALIGN) fprintf(fp, "FF_ALIGN : Alignment directive\n");
    if(flags & FF_3BYTE) fprintf(fp, "FF_3BYTE : 3-byte data\n");
    if(flags & FF_CUSTOM) fprintf(fp, "FF_CUSTOM : Custom data type\n");
    if(flags & FF_IMMD) fprintf(fp, "FF_IMMD : Immediate has value\n");
    if(flags & FF_JUMP) fprintf(fp, "FF_JUMP : Jump has table\n");

    // fprintf(fp, "-------------------------------------------------------------------------------\n");

    // if((sline = GetMnem(ea)) != "") fprintf(fp, "Get instruction name - GetMnem(ea) : %s\n", sline);  // Get instruction name
    // if((sline = GetOpnd(ea, 0)) != "") fprintf(fp, "GetOpnd 1 - GetOpnd(ea, 0): %s\n", sline);  // Get instruction operand 1
    // if((sline = GetOpnd(ea, 1)) != "") fprintf(fp, "GetOpnd 2 - GetOpnd(ea, 1): %s\n", sline);  // Get instruction operand 2
    // if((sline = AltOp(ea, 0)) != "") fprintf(fp, "AltOp 1 - AltOp(ea, 0): %s\n", sline);  // Get manually entered operand 1
    // if((sline = AltOp(ea, 1)) != "") fprintf(fp, "AltOp 2 - AltOp(ea, 1): %s\n", sline);  // Get manually entered operand 2

    if((sline = GetFunctionCmt(ea, 0)) != "") fprintf(fp, "GetFunctionCmt 1 - GetFunctionCmt(ea, 0): %s\n", sline);  // Get Comment 1
    if((sline = GetFunctionCmt(ea, 1)) != "") fprintf(fp, "GetFunctionCmt 2 - GetFunctionCmt(ea, 1): %s\n", sline);  // Get Comment 2

    fprintf(fp, "===============================================================================\n");
  }

  fclose(fp);
}
```

Недокументированные функции скриптов IdaPro

Следует сказать, что скриптовый движок «Иды» имеет несколько недокументированных функций, например:

```
_peek();
_poke();
_lpoke();
_call();
_time();
____();
```

и др.

Особое внимание вызывает последняя функция **____()**, поскольку информации по ней в Интернете практически нет. Путем экспериментов, удалось выяснить, что эта функция то же самое, что и функция **IDA SDK**

```
____() = netnode_altval(RootNode, -2, asc('A'));
```

Описание функции **netnode_altval** находим в документации **IDA SDK:**

```
// Helper function. It should not be called directly!
idaman nodeidx_t ida_export netnode_altval(nodeidx_t num, nodeidx_t alt, char tag);
```

Интересно, «Ида» не рекомендуем нам вызывать эту функцию непосредственно, хотя сама, в данном случае, вызывает эту функцию напрямую. По крайней мере, вызовов такого типа в документации **IDA SDK** нет.

Похоже, что функция **____()** дает адрес отображения своей базы данных в память. Но зачем нужна эта функция в скриптовом движке «Иды» - совершенно непонятно! Ибо в **IDA IDC** практически нет средств для работы с этой памятью.

Первые четыре недокументированные функции вполне хороши для манипуляций с физической памятью, но функция **_call()** не принимает параметров, следовательно, для нее нужно писать обертку-патч где-нибудь в физической памяти, что довольно трудоемко для скриптов. Проще сразу перейти на плагины и не мучатся  . Тем более что в скриптах доступны только порядка **500** функций, а в плагинах – **1500**!

Тем не менее, для примера покажем, прямую реализацию и вызов функции **netnode_altval**(RootNode, -2, asc('A')) в машинных кодах, в физической памяти. Однако непосредственное выполнение ее требует осторожности, так как ваши значения для используемых адресов могут быть другими, что может привести к сбою в работе компьютера.

```
//////////////////////////////////////////////////////////////////////////
// netnode_altval.idc
//////////////////////////////////////////////////////////////////////////
#include <idc.idc>

static main() {
  auto ea, pRootNode, pnetnode_altval;

  ea = 0x0060B100;  // Any free data area
  pRootNode = 0x1015A2B4;  // RootNode pointer, maybe other
  pnetnode_altval = 0x100CF428;  // netnode_altval function pointer, maybe other
  /*
  0060B100   55              PUSH EBP
  0060B101   8BEC            MOV EBP,ESP

  0060B103   6A 41           PUSH 41  ; "A"
  0060B105   6A FE           PUSH -2
  0060B107   A1 B4A21510     MOV EAX,DWORD PTR DS:[IDA.RootNode]  ; MOV EAX,DWORD PTR DS:[1015A2B4]
  ; MOV EAX, 0FF000001
  0060B10C   50              PUSH EAX
  0060B10D   FF15 28F40C10   CALL DWORD PTR DS:[IDA.netnode_altval]  ; CALL DWORD PTR DS:[100CF428]
  ; CALL 83EC8B55

  0060B113   5D              POP EBP
  0060B114   C3              RETN
  0060B115   90              NOP
  */
  _poke(ea, 0x55); ea = ea + 1;
  _poke(ea, 0x8B); _poke(ea + 1, 0xEC); ea = ea + 2;
  _poke(ea, 0x6A); _poke(ea + 1, 0x41); ea = ea + 2;
  _poke(ea, 0x6A); _poke(ea + 1, 0xFE); ea = ea + 2;
  _poke(ea, 0xA1); _lpoke(ea + 1, pRootNode); ea = ea + 5;
  _poke(ea, 0x50); ea = ea + 1;
  _poke(ea, 0xFF); _poke(ea + 1, 0x15); _lpoke(ea + 2, pnetnode_altval); ea = ea + 6;
  _poke(ea, 0x5D); ea = ea + 1;
  _poke(ea, 0xC3); ea = ea + 1;
  _poke(ea, 0x90); ea = ea + 1;

  Message("\n____() = 0x%08X\n", ____());
  Message("\n_call(0x%08X) = 0x%08X\n", ea, _call(ea));
}
```

Разбор узла дизассемблерного листинга IdaPro с помощью плагинов

Перейдем теперь к собственно полученному решению, как не трудно догадаться, с помощью плагинов.

За основу мы взяли плагин **getlines.cpp** из **IDA SDK**. Кто бы мог подумать, что он там есть? А я облазил весь Интернет, в поисках прототипа .

Вот полученный результат:

```
//-------------------------------------------------------------------------
// Code for SaveListing.plw plugin
// It demonstrates how to get the disassembly lines for all addresses
//-------------------------------------------------------------------------

// Bounden declarations
#define __NT__
#define __IDP__

#include <ida.hpp>
#include <idp.hpp>
#include <bytes.hpp>
#include <loader.hpp>
#include <kernwin.hpp>
#include <expr.hpp>

//-------------------------------------------------------------------------
// init
//-------------------------------------------------------------------------
int init(void) {
  return PLUGIN_OK;
}

//-------------------------------------------------------------------------
// term
//-------------------------------------------------------------------------
void term(void) {
}

//-------------------------------------------------------------------------
// run
//-------------------------------------------------------------------------
void run(int /*arg*/) {
  char *outfile = askfile_c(true, "*.lst", "Please specify the output file:");
  FILE *fp = fopen(outfile, "w");

  if(fp == NULL) {
    warning("Cannot create output file!");
    return;
  }

  //*** Structures ***

  // fprintf(fp, "================================================================================================================\n");
  // fprintf(fp, "\n");

  linearray_t linar(0);

  int i;
  int line_qty = 0;

  structplace_t st_pl(0, 0, 0);
  linar.set_place(&st_pl);

  while(true) {
    for(i = 0; i < linar.get_linecnt(); i++) {  // Process all of them
      char *line = linar.down();  // Get line
      char buf[MAXSTR];

      tag_remove(line, buf, sizeof(buf));  // Remove color codes

      // msg("%d: %s\n", i, buf);  // Display it on the message window
```

```
    fprintf(fp, "%s\n", buf);

    line_qty++;
  }

  if(line_qty%1000 == 0)
     msg("%d lines of structures done...\n", line_qty);

  st_pl.next(0);

  if(st_pl.ending(0)) {
    char *line = linar.down();  // Get line
    char buf[MAXSTR];

    tag_remove(line, buf, sizeof(buf));  // Remove color codes

    // msg("%d: %s\n", i, buf);  // Display it on the message window
    fprintf(fp, "%s\n", buf);

    line_qty++;

    break;
  }
}

// fprintf(fp, "=================================================================================================\n");
fprintf(fp, "\n");

msg("\n%d lines of structures has written!\n\n", line_qty);

//*** Code & data ***

ea_t ea;
ea_t min_ea = inf.minEA;
ea_t max_ea = inf.maxEA;

int n, flags;
int j = 0;

for(ea = min_ea; ea < max_ea && ea != BADADDR; ea = next_not_tail(ea)) {  // && ea < min_ea + 100;
  idaplace_t pl;

  pl.ea = ea;
  pl.lnnum = 0;

  // User defined data for linearray_t: int *flag
  // *flag value is a combination of:
  // #define IDAPLACE_HEXDUMP 0x000F  // produce hex dump
  // #define IDAPLACE_STACK   0x0010  // produce 2/4/8 bytes per undefined item
  // (used to display the stack contents)
  // the number of displayed bytes depends on the stack bitness
  // not used yet because it confuses users:
  ////#define IDAPLACE_SHOWPRF 0x0020  // display line prefixes
  // #define IDAPLACE_SEGADDR 0x0040  // display line prefixes with the segment part
  flags = calc_default_idaplace_flags();

  linearray_t ln(&flags);

  ln.set_place(&pl);

  // msg("Printing disassembly lines:\n");

  n = ln.get_linecnt();  // How many lines for this address?

  for(i = 0; i < n; i++) {  // Process all of them
    char *line = ln.down();  // Get line
    char buf[MAXSTR];
```

```
      tag_remove(line, buf, sizeof(buf));  // Remove color codes

      // msg("%d: %s\n", i, buf);  // Display it on the message window
      fprintf(fp, "%s\n", buf);

      j++;
    }

    if(j%1000 == 0)
      msg("%d lines done...\n", j);
  }

  msg("\n\n%d lines has written! Output file is done.\n", j);

  fclose(fp);
}

//--------------------------------------------------------------------------

char comment[] = "Generate disassembly lines for all addresses";
char help[] = "Generate disassembly lines for all addresses\n";

//--------------------------------------------------------------------------

// This is the preferred name of the plugin module in the menu system
// The preferred name may be overridden in plugins.cfg file
char wanted_name[] = "Save the Listing";

//--------------------------------------------------------------------------

// This is the preferred hotkey for the plugin module
// The preferred hotkey may be overriden in plugins.cfg file
// Note: IDA won't tell you if the hotkey is not correct
// It will just disable the hotkey.
char wanted_hotkey[] = "";

//--------------------------------------------------------------------------
// PLUGIN DESCRIPTION BLOCK
//--------------------------------------------------------------------------
plugin_t PLUGIN =
{
  IDP_INTERFACE_VERSION,
  0,  // Plugin flags
  init,  // Initialize
  term,  // Terminate. this pointer may be NULL
  run,  // Invoke plugin
  comment,  // Long comment about the plugin it could appear in the status line or as a hint
  help,  // Multiline help about the plugin
  wanted_name,  // The preferred short name of the plugin
  wanted_hotkey  // The preferred hotkey to run the plugin
};

//--------------------------------------------------------------------------
//--------------------------------------------------------------------------
```

Этот код сохраняет структуры и дизассемблерный листинг в выбранный файл. Перечисления не обрабатываются, но могут быть добавлены по аналогии со структурами.

Вы можете компилировать этот файл обычными способами, но для разнообразия я использовал командный файл **plw.cmd**. Выставьте только свои параметры окружения. Да, и не забудьте про **ida.lib** из **SDK**. У меня он находится уровнем выше (чтобы быть общим для разных плагинов). Если у вас его нет, то сгенерите его сами из **ida.wll**, по технологии, описанной в моей последней статье. Правда, где вы возьмете отладочные символы для этого файла, я не знаю, разве, что из самого **ida.lib**  .

```
::*** Командный файл plw.cmd
::*** Параметры компиляции:

SET PlugName=plugin

SET CL_EXE="C:\Program Files\Microsoft Visual Studio\VC98\Bin\cl.exe"
SET LINK_EXE="C:\Program Files\Microsoft Visual Studio\VC98\Bin\link.exe"
SET IncVC98="C:\Program Files\Microsoft Visual Studio\VC98\Include"
SET LibVC98="C:\Program Files\Microsoft Visual Studio\VC98\Lib"
SET IncIdaSdk="D:\MyDocs\IdaSdk\include"

%CL_EXE% /I%IncVC98% /I%IncIdaSdk% -Gz /O2 /MD /GX /W3 /FD /nologo /c %PlugName%.cpp > a.a

%LINK_EXE% /MACHINE:I386 /NOLOGO /INCREMENTAL:NO /NODEFAULTLIB:"LIBC" /subsystem:windows /dll /out:%PlugName%.plw
/def:%PlugName%.def %PlugName%.obj ..\ida.lib /LIBPATH:%LibVC98% > a.b

del *.obj
del *.exp
del *.idb
del %PlugName%.lib

:: pause
```

Еще нужен файл **plugin.def**, общий для всех проектов:

```
LIBRARY plugin
EXPORTS
PLUGIN
```

Если все сделаете правильно, то получите плагин **plugin.plw**. Я его просто переименовал в **SaveListing.plw**. Надеюсь, вы знаете, как пользоваться плагинами  . Но можно вызвать данный плагин и с помощью скрипа:

```
RunPlugin("SaveListing", 0);
```

(точку с запятой можно и не писать ).

По умолчанию, «Ида» ищет плагины в папке **plugins**.

Если в файле **plugins.cfg** «Иды» добавить строчку

```
Save_the_listing    SaveListing    Ctrl-[    0    ; Save disassembler listing
```

то вызов нашего плагина можно будет также осуществить по «горячим» клавишам **Ctrl+[** . Вы можете определить свою комбинацию клавиш.

Выводы

Хочется надеяться, что эта статья не послужит поводом для **Ильфака Гильфанова** закрыть в его демо-версиях возможность использования скриптов и плагинов.

Очевидно, что использование плагинов существо более эффективно, чем применение скриптов практически при том же уровне сложности работы. Конечно для этого нужно иметь **IDA SDK** последних версий и компилятор **C++**. Но что нам мешает применить Интернет в личных целях  ? Эффективность плагинов заставляет пересмотреть наше отношение к внешним скриптам. Можно попытаться с их помощью проделать туже работу, что и в первых трех статьях и даже автоматизировать процесс восстановления бинарного кода программ (на уровне ассемблера) настолько, насколько это возможно. Так что для будущих статей темы всегда найдутся  .

Примечание

К данному тексту приложен файл **SaveListing.004** (http://erfaren.narod.ru/Asm/SaveListing.004 - измените расширение в **zip**), с результатами исследования и сгенерированным плагином. Можно также посмотреть **html** версию этой статьи (http://erfaren.narod.ru/Asm/Erfaren004.htm) либо ее **pdf**-файл (http://erfaren.narod.ru/Asm/Erfaren-004-Save-Listing-IdaPro6-demo.pdf).