|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Министерство науки и высшего образования Российской Федерации  Федеральное государственное бюджетное образовательное учреждение  высшего образования  «Владимирский государственный университет  имени Александра Григорьевича и Николая Григорьевича Столетовых»  (ВлГУ)  разработка компилятора подмножества  процедурно-ориентированного языка  Пояснительная записка  RU. 643.02068048.0001-01 81 01  На 11 листах   |  |  |  | | --- | --- | --- | | Руководитель |  | к.т.н. доцент кафедры ИЗИ Ю.М. Монахов | |  | | Исполнитель |  | студент гр. ИСБ-117 А.И. Кондратьева |   **Владимир 2020** |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
| *Инв.№ подл.* |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **АННОТАЦИЯ**  В данном программном документе приведён текст компилятора подмножества процедурно-ориентированного языка. Текст программы реализован на языке python в интегрированной среде разработки PyCharm Edu 2020.1.1. Основная функция компилятора — это проверка исходной цепочки символов на принадлежность к входному языку и генерация выходной цепочки символов на языке машинных команд или ассемблере.  Можно выделить основные части компилятора подмножества процедурно-ориентированного языка:  1) лексический анализатор  2) синтаксический анализатор  3) Оптимизатор  4) Генератор кода | | | | | | | | | | |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
|  |  |  |  |  | *643-02068048-00001-01* | | | | | |
|  |  |  |  |  |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |
| *Инв.№ подл.* |  | *Разраб.* | |  |  |  | *«Разработка компилятора подмножества процедурно-ориентированного языка»* | *Лит.* | | | *Лист* | *Листов* |
| *Пров.* | |  |  |  |  |  |  | *2* |  |
|  | |  |  |  | *ИСБ-117, А.И. Кондратьева* | | | | |
| *Н.контр.* | |  |  |  |
| *Утв.* | |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **СОДЕРЖАНИЕ**  СОДЕРЖАНИЕ 3  1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА 4  1.1 ОСНОВНЫЕ ТРЕБОВАНИЯ 4  1.2 СОЗДАНИЕ ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА 4  1.3 РАЗРАБОТКА СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА 7  1.4 ПОСТРОЕНИЕ ОПТИМИЗАЦИИ 10  1.5 ГЕНЕРАЦИЯ ОБЪЕКТНОГО КОДА 11  2 ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ 15  2.1 СОБЛЮДЕНИЕ ТРЕБОВАНИЙ 15 | | | | | | |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
| *Инв.№ подл.* |  |
|  |  |  |  |  | *643-02068048-00001-01* | *Лист* |
|  |  |  |  |  |
| *3* |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |
|  | | **1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА**  **1.1 Основные требования**  Разработка будет производиться в соответствии со следующими требованиями:   * Требования к входному языку:  1. Должны присутствовать операторные скобки; 2. Должна игнорироваться идентация программы; 3. Должны поддерживаться комментарии любой длины; 4. Входная программа должна представлять собой единый модуль, но поддерживать вызов функций.  * Требования к операторам:  1. Оператор присваивания; 2. Арифметические, логические операторы; 3. Условный оператор; 4. Операторы цикла (break, continue); 5. Базовый вывод; 6. Типы (целочисленный, вещественный).  * Требования к выходному языку:  1. Ассемблер для архитектуры x86.    1. **Создание лексического анализатора**   Лексический анализатор является первой фазой работы компилятора. Работа лексического анализатора основана на регулярных выражениях. На вход подаются строки исходной программы. В самом анализаторе присутствует набор функций для получения результата анализа строки. В результате работы лексера получается первичное дерево. Для лексического анализатора и парсера используется RPLY. Это библиотека Python с теми же лексическими и парсинговыми инструментами, но с более качественным API.  В Листинге 1 приведен список регулярных выражений. | | | | | | |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
| *Инв.№ подл.* |  |
|  |  |  |  |  | *643-02068048-00001-01* | *Лист* |
|  |  |  |  |  |
| *4* |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Листинг 1 – Список регулярных выражений  self.lexer.add('LITERAL', r'"[^\"]\*"')  self.lexer.add('AND', r'and')  self.lexer.add('NOT', r'not')  self.lexer.add('OR', r'or')  self.lexer.add('IF', r'if')  self.lexer.add('THEN', r'then')  self.lexer.add('ELSE', r'else')  self.lexer.add('WHILE', r'while')  self.lexer.add('DO', r'do')  self.lexer.add('BREAK', r'break')  self.lexer.add('CONTINUE', r'continue')  self.lexer.add('BEGIN', r'\{')  self.lexer.add('END', r'\}')  self.lexer.add('INTEGER', r'integer')  self.lexer.add('FLOAT', r'float')  self.lexer.add('FUNCTION', r'function')  self.lexer.add('VAR', r'var')  self.lexer.add('PROGRAM', r'program')  self.lexer.add('LEQUAL', r'\<=')  self.lexer.add('GEQUAL', r'\>=')  self.lexer.add('NOT\_EQUAL', r'\!=')  self.lexer.add('EQUALS', r'\:=')  self.lexer.add('PRINT', r'print')  self.lexer.add('OPEN\_PAREN', r'\(')  self.lexer.add('CLOSE\_PAREN', r'\)')  self.lexer.add('SEMI\_COLON', r'\;')  self.lexer.add('COLON', r'\:')  self.lexer.add('COMMA', r'\,')  self.lexer.add('EQUAL', r'\=')  self.lexer.add('GTHAN', r'\>')  self.lexer.add('LTHAN', r'\<')  self.lexer.add('SUM', r'\+')  self.lexer.add('SUB', r'\-')  self.lexer.add('MUL', r'\\*')  self.lexer.add('DIV', r'\/')  self.lexer.add('NUMBER', r'[0-9]+(\.[0-9]+)?')  self.lexer.add('ID', r'[a-zA-Z]\*')  self.lexer.ignore(r'#[^\#]\*#')  self.lexer.ignore('\s+')  def get\_lexer(self):  self.\_add\_tokens()  return self.lexer.build()  Листинг 2 – Работа лексического анализатора  fname = "program.txt" with open(fname) as f:  text\_input = f.read()  lexer = Lexer().get\_lexer() tokens = lexer.lex(text\_input) new\_tokens = lexer.lex(text\_input)  token\_stream=[] for t in new\_tokens:  token\_stream.append(t)  print(t) *#print(token\_stream)* | | | | | | |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
| *Инв.№ подл.* |  |
|  |  |  |  |  | *643-02068048-00001-01* | *Лист* |
|  |  |  |  |  |
| *5* |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | На рисунке 1 изображена символьная таблица – результат работы лексического анализатора.  Token('WHILE', 'while')  Token('OPEN\_PAREN', '(')  Token('ID', 'n')  Token('GTHAN', '>')  Token('NUMBER', '1')  Token('CLOSE\_PAREN', ')')  Token('COLON', ':')  Token('BEGIN', '{')  Token('ID', 'temp')  Token('EQUALS', ':=')  Token('ID', 'res')  Token('SEMI\_COLON', ';')  Token('ID', 'res')  Token('EQUALS', ':=')  Token('ID', 'temp')  Token('MUL', '\*')  Token('ID', 'n')  Token('SEMI\_COLON', ';')  Token('ID', 'remp')  Token('EQUALS', ':=')  Token('ID', 'n')  Token('SEMI\_COLON', ';')  Token('ID', 'n')  Token('EQUALS', ':=')  Рисунок 1 – Символьная таблица   * 1. **Разработка синтаксического анализатора**   Вторая фаза компилятора – это синтаксический анализ. На вход синтаксического анализатора подаются строки из первичного дерева. Данный компонент принимает список токенов на входе и создает на выходе абстрактное синтаксическое дерево (АСД). Как было описано выше, здесь используется библиотека RPLY. При помощи двух компонентов парсера и РБНФ создаём работающий компилятор, интерпретирующий язык. В Листинге 3 приведен код парсера.  Листинг 3- Код синтаксического анализатора  class Parser():  def \_\_init\_\_(self, module, builder, printf, struct):  self.pg = ParserGenerator(  ['AND', 'NOT', 'OR', 'IF', 'ELSE',  'WHILE', 'BREAK', 'CONTINUE', 'BEGIN', 'END',  'INTEGER', 'FLOAT', 'FUNCTION', 'VAR', 'PROGRAM',  'PRINT', 'OPEN\_PAREN', 'CLOSE\_PAREN', 'SEMI\_COLON',  'COLON', 'COMMA', 'EQUAL', 'EQUALS', 'NOT\_EQUAL',  'GTHAN', 'GEQUAL', 'LTHAN', 'LEQUAL',  'SUM', 'SUB', 'MUL', 'DIV',  'NUMBER', 'ID', 'LITERAL'  ],  precedence=[  ('left',['SUM','SUB']),  ('left',['DIV','MUL'])  ]  ) | | | | | | |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
| *Инв.№ подл.* |  |
|  |  |  |  |  | *643-02068048-00001-01* | *Лист* |
|  |  |  |  |  |
| *5* |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | self.module = module  self.builder = builder  self.printf = printf  self.struct = struct   def parse(self):   @self.pg.production('program : head globalp functions main')  def program(p):  return p[3]    @self.pg.production('head : PROGRAM ID SEMI\_COLON')  def head(p):  pass    @self.pg.production('globalp : declrs')  def globalp1(p):  return p[0]    @self.pg.production('globalp : declrs globalp')  def globalp2(p):  return p[0],p[1]   @self.pg.production('declrs : VAR ids COLON type SEMI\_COLON')  def declrs(p):  return p[1]   @self.pg.production('functions : function')  def functions2(p):  return p[0].eval()    @self.pg.production('functions : function functions')  def functions2(p):  return Eval\_(self.builder,self.module,p[0],p[2])    @self.pg.production('function : FUNCTION ID OPEN\_PAREN globalp CLOSE\_PAREN COLON type SEMI\_COLON BEGIN stmts END')  def function(p):  return Func\_(self.builder,self.module,p[1],p[3],p[9])  @self.pg.production('ids : ID') def ids1(p):  return p[0]  @self.pg.production('ids : ID COMMA ids') def ids2(p):  return p[0],p[2]   @self.pg.production('type : INTEGER') @self.pg.production('type : FLOAT') def type(p):  return p[0]  @self.pg.production('main : BEGIN stmts END') def main(p):  return p[1]  @self.pg.production('stmts : stmt SEMI\_COLON') def stmts1(p):  return p[0] | | | | | | |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
| *Инв.№ подл.* |  |
|  |  |  |  |  | *643-02068048-00001-01* | *Лист* |
|  |  |  |  |  |
| *6* |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | @self.pg.production('stmts : stmt SEMI\_COLON stmts') def stmts2(p):  return Eval\_(self.builder,self.module,p[0],p[2])  @self.pg.production('stmts : BREAK SEMI\_COLON stmts') def break\_(p):  global goto  goto="break"  return p[2]  @self.pg.production('stmts : CONTINUE SEMI\_COLON stmts') def continue\_(p):  global goto  goto="continue"  return p[2]  @self.pg.production('stmt : ID EQUALS expression') def equals\_(p):  return Store\_(self.builder,self.module, self.struct, p[0], p[2])  @self.pg.production('stmt : IF OPEN\_PAREN bool CLOSE\_PAREN COLON BEGIN stmts END') def if\_stmt(p):  return If\_(self.builder,self.module, p[2], p[6])  @self.pg.production('stmt : IF OPEN\_PAREN bool CLOSE\_PAREN COLON BEGIN stmts END ELSE BEGIN stmts END') def ifelse\_stmt(p):  return Ifelse\_(self.builder,self.module, p[2], p[6], p[10])  @self.pg.production('stmt : WHILE OPEN\_PAREN bool CLOSE\_PAREN COLON BEGIN stmts END') def for\_stmt(p):  return While\_(self.builder,self.module,p[2],p[6])   @self.pg.production('stmt : PRINT OPEN\_PAREN expression CLOSE\_PAREN') def print\_stmt(p):  return Print(self.builder,self.module, self.printf, p[2])  @self.pg.production('stmt : PRINT OPEN\_PAREN LITERAL CLOSE\_PAREN') def print\_literal(p):  return Literal\_(self.builder,self.module, self.printf, p[2])   @self.pg.production('bool : expression EQUAL expression') @self.pg.production('bool : expression GTHAN expression') @self.pg.production('bool : expression LTHAN expression') @self.pg.production('bool : expression GEQUAL expression') @self.pg.production('bool : expression LEQUAL expression') @self.pg.production('bool : expression NOT\_EQUAL expression') def bool\_compare(p):  left = p[0]  right = p[2]  operator = p[1]  if operator.gettokentype() == 'EQUAL':  return Equal(self.builder,self.module, left, right)  elif operator.gettokentype() == 'GTHAN':  return Gthan(self.builder,self.module, left, right)  elif operator.gettokentype() == 'LTHAN':  return Lthan(self.builder,self.module, left, right)  elif operator.gettokentype() == 'GEQUAL':  return Gequal(self.builder,self.module, left, right) | | | | | | |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
| *Инв.№ подл.* |  |
|  |  |  |  |  | *643-02068048-00001-01* | *Лист* |
|  |  |  |  |  |
| *6* |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | elif operator.gettokentype() == 'LEQUAL':  return Lequal(self.builder,self.module, left, right)  elif operator.gettokentype() == 'NOT\_EQUAL':  return Not\_equal(self.builder,self.module, left, right)  @self.pg.production('bool : bool AND bool') def and\_bool(p):  return And\_(self.builder,self.module, p[0], p[2]) @self.pg.production('bool : bool OR bool') def or\_bool(p):  return Or\_(self.builder,self.module, p[0], p[2]) @self.pg.production('bool : NOT bool') def not\_bool(p):  return Not\_(self.builder,self.module, p[1]) @self.pg.production('bool : OPEN\_PAREN bool CLOSE\_PAREN') def paren\_bool(p):  return p[1]  @self.pg.production('expression : expression SUM expression') @self.pg.production('expression : expression SUB expression') @self.pg.production('expression : expression MUL expression') @self.pg.production('expression : expression DIV expression') def expression(p):  left = p[0]  right = p[2]  operator = p[1]  if operator.gettokentype() == 'SUM':  return Sum(self.builder,self.module, left, right)  elif operator.gettokentype() == 'SUB':  return Sub(self.builder,self.module, left, right)  elif operator.gettokentype() == 'MUL':  return Mul(self.builder,self.module, left, right)  elif operator.gettokentype() == 'DIV':  return Div(self.builder,self.module, left, right)  @self.pg.production('expression : NUMBER') def number(p):  return Number(self.builder,self.module, p[0].value) @self.pg.production('expression : SUB NUMBER') def number(p): | | | | | | |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
| *Инв.№ подл.* |  |
|  |  |  |  |  | *643-02068048-00001-01* | *Лист* |
|  |  |  |  |  |
| *6* |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | return Number(self.builder,self.module, str(0 - int(p[1].value)))  @self.pg.production('expression : OPEN\_PAREN expression CLOSE\_PAREN')  def paren\_expr(p):  return p[1]   @self.pg.production('expression : ID')  def id\_expr(p):  return Load\_(self.builder,self.module, p[0])    @self.pg.production('expression : ID OPEN\_PAREN ids CLOSE\_PAREN')  def function\_expr(p):  return Call\_(self.builder, self.module,p[0],p[2])   @self.pg.error  def error\_handle(token):  raise ValueError(token)  def get\_parser(self):  return self.pg.build()  На рисунке 2 показан результат работы синтаксического анализатора.  a integer  b integer  c integer  i integer  remp integer  res integer  temp integer  u float  Max integer  fir integer  sec integer  Рисунок 2 – АСД  **1.4 Построение оптимизатора**  Построение оптимизации заключается в удалении пустых функций. Проход по всему первоначальному дереву. На листинге 4 приведен код оптимизатора.  Листинг 4 – Код функции оптимизации  *#OPTIMIZATOR1* global new\_module new\_module=self.module for x in range(0,len(value\_num)):  if value\_num[x]:  TryVal=value\_num[x]  *#Strip* if hasNumbers(TryVal):  TryVal=TryVal[:-1]  if hasNumbers(TryVal):  TryVal=TryVal[:-1]  Object = re.search(r'store[^\@]\*@"'+TryVal+'"', str(self.module))  if Object:  print(Object.group())  Object = re.search(r'load[^\@]\*@"'+TryVal+'"', str(self.module))  if Object:  print(Object.group())  else:  print(TryVal," Not Found!") | | | | | | |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
| *Инв.№ подл.* |  |
|  |  |  |  |  | *643-02068048-00001-01* | *Лист* |
|  |  |  |  |  |
| *7* |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | new\_module=re.sub(r'store[^\@]\*@"'+TryVal+'"', "", str(new\_module))  new\_module=re.sub('@"'+TryVal+'" = internal global i32 undef', "", str(new\_module))  for y in range(1,count[x]):  TryVal=TryVal+str(y+1)  Object = re.search(r'store[^\@]\*@"'+TryVal+'"', str(self.module))  if Object:  print(Object.group())  Object = re.search(r'load[^\@]\*@"'+TryVal+'"', str(self.module))  if Object:  print(Object.group())  else:  print(TryVal," Not Found!")  new\_module=re.sub(r'store[^\@]\*@"'+TryVal+'"', "", str(new\_module))  new\_module=re.sub('@"'+TryVal+'" = internal global i32 undef', "", str(new\_module))  *#Strip* if hasNumbers(TryVal):  TryVal=TryVal[:-1]  if hasNumbers(TryVal):  TryVal=TryVal[:-1] *#OPTIMIZATOR2* Object = re.search(r'call i32 @"function"', str(new\_module)) if not Object:  print('Deleting function block ...')  new\_module=re.sub(r'define i32 @"function"[^\}]\*}', "", str(new\_module))   * 1. **Генератор объектного кода**   Последняя часть компилятора – это генератор кода. Его роль заключается в том, чтобы преобразовывать АСД в машинный код или промежуточное представление. Здесь будет происходить преобразование АСД в промежуточное представление LLVM (LLVM IR).  Листинг 5 – Код генератора объектного кода  **class** CodeGen():  **def** \_\_init\_\_(self):  self.binding = binding  self.binding.initialize()  self.binding.initialize\_native\_target()  self.binding.initialize\_native\_asmprinter()  self.\_config\_llvm()  self.\_create\_execution\_engine()  self.\_declare\_print\_function()   **def** \_config\_llvm(self):  *#Config* **global** base\_func  self.module = ir.Module(name=\_\_file\_\_)  self.module.triple = self.binding.get\_default\_triple()    func\_type = ir.FunctionType(ir.VoidType(), [], **False**)  base\_func = ir.Function(self.module, func\_type, name=**"main"**)  block = base\_func.append\_basic\_block(name=**"entry"**)  self.builder = ir.IRBuilder(block) | | | | | | |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
| *Инв.№ подл.* |  |
|  |  |  |  |  | *643-02068048-00001-01* | *Лист* |
|  |  |  |  |  |
| *8* |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **def** \_create\_execution\_engine(self):  *#Engine* target = self.binding.Target.from\_default\_triple()  target\_machine = target.create\_target\_machine()  backing\_mod = binding.parse\_assembly(**""**)  engine = binding.create\_mcjit\_compiler(backing\_mod, target\_machine)  self.engine = engine  **def** \_declare\_print\_function(self):  *#Printf* voidptr\_ty = ir.IntType(8).as\_pointer()  printf\_ty = ir.FunctionType(ir.IntType(32), [voidptr\_ty], var\_arg=**True**)  printf = ir.Function(self.module, printf\_ty, name=**"printf"**)  self.printf = printf  **def** \_compile\_ir(self):  *#Compile* self.builder.ret\_void()      Рисунок 3 – Код программы Рисунок 4 – Код на LLVM | | | | | | |
| *Подп. И дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. И дата* |  |
| *Инв.№ подл.* |  |
|  |  |  |  |  | *643-02068048-00001-01* | *Лист* |
|  |  |  |  |  |
| *9* |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **2 ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ**  **2.1 Соблюдение требований**  При проектировании компилятора к основному языку были установлены следующие требования: наличие операторных скобок, игнорирование пробелов и идентации программы, поддержка многострочных комментариев и вызова функций. Наличие операторов присваивания, условных, цикла, break-continue, арифметических, логических. Должны присутствовать два типа данных. И выходная программа должна быть на ассемблере.  К сожалению, я не смогла запустить файл output.ll через командную строку, следовательно, проверить машинный код – тоже. | | | | | | |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
| *Инв.№ подл.* |  |
|  |  |  |  |  | *643-02068048-00001-01* | *Лист* |
|  |  |  |  |  |
| *10* |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |
|  | |  | | | | | | | |
| *Подп. и дата* |  |
| *Инв.№ дубл.* |  |
| *Взам.инв.№* |  |
| *Подп. и дата* |  |
| *Инв.№ подл.* |  |
|  |  |  |  |  | *643-02068048-00001-01* | *Лист* | |
|  |  |  |  |  |
| *11* | |
| *Изм* | *Лист* | *№ докум.* | *Подп.* | *Дата* |

Реквизиты к курсовой работе Кондратьевой Алёны Игоревны.

