Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
Кафедра Вычислительных систем (ВС)

ОТЧЕТ
о практической работе
«Реализация простого конвертера с подмножества Python
на Си»
Уровень: «Отлично»

Работу выполнил:
студент 1 курса
группы ИС-541
Устюжанин Д.К.

г. Новосибирск
2025 г.

Содержание

Введение 3

Исходный код конвертера на Си 4

Листинг 1.1 – файл consts.h 4

Листинг 2.1 – файл strfunc.h 5

Листинг 2.2 – файл strfunc.c 7

Листинг 3.1 – файл conpyc.h 12

Листинг 3.2 – файл conpyc.c 13

Листинг 4.1 – файл main.c 18

Описание алгоритма распознавания конструкций 21

Примеры работы 22

Хороший пример (работает корректно) 22

Плохой пример (работает корректно) 23

Вывод 24

Введение

Цель работы: Реализовать один из вариантов транслирования, конвертер с языка программирования Python на язык программирования Си.

Исходный код конвертера на Си

Листинг 1.1 – файл consts.h

```
#include <wchar.h>

#define BUFFER_SIZE 1000 // max buffer size
#define WORD_SIZE 100 // max variable size
#define WORDS_SIZE 100 // max variables array size
```

Листинг 2.1 – файл strfunc.h

```
#include <stdio.h>
#include "consts.h"

// find word in words array
wchar_t *find_word_a(wchar_t words[WORDS_SIZE][WORD_SIZE], wchar_t
word[WORD_SIZE]);

// add (copy) word in words array at specific index
wchar_t *add_word_a(wchar_t words[WORDS_SIZE][WORD_SIZE],
    wchar_t word[WORD_SIZE], size_t index);

// compare two words
wchar_t *cmpnw(wchar_t *word1, wchar_t *word2, size_t n);

// checks if char is space-char
_Bool wisspace(wchar_t wch);

// skip cpecific char
size_t *skipwch(wchar_t buffer[BUFFER_SIZE], wchar_t ch, size_t
*i);

// skip space-char
size_t *skipwspace(wchar_t buffer[BUFFER_SIZE], size_t *i);

// skip before cpecific char
size_t *skip_before_wch(wchar_t buffer[BUFFER_SIZE], wchar_t ch,
size_t *i);

// skip space-char
```

```
size_t *skip_before_wspace(wchar_t buffer[BUFFER_SIZE], size_t
*i);

// put before cpecific char
size_t *put_before_wch(wchar_t buffer[BUFFER_SIZE], wchar_t ch,
size_t *i);

// put space-char
size_t *put_before_wspace(wchar_t buffer[BUFFER_SIZE], size_t *i);

// insert wide-word to stdout
void winsw_out(wchar_t word[WORD_SIZE]);

// copy one wide-word to another
size_t wcpy(wchar_t src[WORD_SIZE], wchar_t dest[WORD_SIZE]);

// copy wide-word from stdin to another and store it
size_t wcpy_wrd_buf(wchar_t buffer[BUFFER_SIZE], wchar_t
word[WORD_SIZE], size_t *i);
```

Листинг 2.2 – файл strfunc.c

```
#include "strfunc.h"

// find word in words array
wchar_t *find_word_a(wchar_t words[WORDS_SIZE][WORD_SIZE], wchar_t
word[WORD_SIZE]) {
  for (size_t i=0; i<WORDS_SIZE; i++)
    for (size_t j=0; j<WORD_SIZE; j++) {
      if (*(*(words+i)+j) != *(word+j)) break;
      if (*(*(words+i)+j) == L'\0' && *(word+j) == L'\0') return
*(words+i);
      if (*(*(words+i)+j) == L'\0') break;
    }
  return NULL;
}

// add (copy) word in words array at specific index
wchar_t *add_word_a(wchar_t words[WORDS_SIZE][WORD_SIZE], wchar_t
word[WORD_SIZE], size_t index) {
  if (index >= WORDS_SIZE) return NULL;
  for (size_t i=0; i<WORD_SIZE; i++) {
    *(*(words+index)+i) = *(word+i);
    if (*(word+i) == '\0') break;
  }
  *(*(words+index)+WORD_SIZE-1) = L'\0';
  return *(words+index);
}

// compare two words
wchar_t *cmpnw(wchar_t *word1, wchar_t *word2, size_t n) {
  if (!word1 || !word2) return NULL;
```

```
  for (size_t i=0; i<n; i++) {
    if (*(word1+i) != *(word2+i)) return NULL;
    if (*(word1+i) == L'\0' && *(word2+i) == L'\0') break;
    if (*(word1+i) == L'\0' || *(word2+i) == L'\0') return NULL;
  }
  return (wchar_t*)word1;
}

// checks if char is space-char
_Bool wisspace(wchar_t wch) {
  if (wch == L'\n' || wch == L'\t' || wch == L' ' || wch == L'\r')
return 1;
  return 0;
}

// skip cpecific char
size_t *skipwch(wchar_t buffer[BUFFER_SIZE], wchar_t ch, size_t
*i) {
  for (; (*i) < BUFFER_SIZE &&
    *(buffer+(*i)) == ch && *(buffer+(*i)) != L'\0'; (*i)++);
  return i;
}

// skip space-char
size_t *skipwspace(wchar_t buffer[BUFFER_SIZE], size_t *i) {
  for (; (*i)<BUFFER_SIZE && wisspace(*(buffer+(*i))); (*i)++);
  return i;
}



// skip before cpecific char
```

```
size_t *skip_before_wch(wchar_t buffer[BUFFER_SIZE], wchar_t ch,
size_t *i) {
  for (; (*i) < BUFFER_SIZE &&
    *(buffer+(*i)) != ch && *(buffer+(*i)) != L'\0'; (*i)++);
  return i;
}

// skip space-char
size_t *skip_before_wspace(wchar_t buffer[BUFFER_SIZE], size_t *i)
{
  for (; (*i)<BUFFER_SIZE && !wisspace(*(buffer+(*i))); (*i)++);
  return i;
}

// put before cpecific char
size_t *put_before_wch(wchar_t buffer[BUFFER_SIZE], wchar_t ch,
size_t *i) {
  for (; (*i) < BUFFER_SIZE &&
    *(buffer+(*i)) != ch && *(buffer+(*i)) != L'\0'; (*i)++)
    putwchar(*(buffer+(*i)));
  return i;
}

// put space-char
size_t *put_before_wspace(wchar_t buffer[BUFFER_SIZE], size_t *i)
{
  for (; (*i)<BUFFER_SIZE && !wisspace(*(buffer+(*i))); (*i)++)
    putwchar(*(buffer+(*i)));
  return i;
}
```

```
// insert wide-word to stdout
void winsw_out(wchar_t word[WORD_SIZE]) {
  for (size_t i=0; i<WORD_SIZE && *(word+i) != L'\0'; i++) {
    putwchar(*(word+i));
  }
}

// copy one wide-word to another
size_t wcpy(wchar_t src[WORD_SIZE], wchar_t dest[WORD_SIZE]) {
  size_t cnt=0;
  for (size_t i=0; i<WORD_SIZE && *(src+i)!=L'\0'; i++) {
    *(dest+i) = *(src+i);
    cnt++;
  }
  if (cnt < WORD_SIZE) *(dest+cnt) = L'\0';
  return cnt;
}

// copy wide-word from stdin to another and store it
size_t wcpy_wrd_buf(wchar_t buffer[BUFFER_SIZE], wchar_t
word[WORD_SIZE], size_t *i) {
  size_t cnt=0; // counts chars
  wchar_t ch=0;
  // skip leading whitespaces
  skipwspace(buffer, i);

  // read word
  for (; (*i)<BUFFER_SIZE && cnt < WORD_SIZE-1; (*i)++) {
    ch = *(buffer+(*i));
    if (wisspace(ch) || ch==L':' || ch==L'(') {*(word+cnt)=L'\0';
return cnt;}
```

```
    *(word+cnt) = ch;
    cnt++;
  }

  // end word as string
  if (cnt < WORD_SIZE) *(word+cnt) = L'\0';

  return cnt;
}
```

Листинг 3.1 – файл conpyc.h

```
#include "consts.h"

// for loop condition (making brackets) processing
void for_cond_proc(wchar_t buffer[BUFFER_SIZE], size_t *i);

// while loop and if condition (making brackets) processing
void whif_cond_proc(wchar_t buffer[BUFFER_SIZE], size_t *i);

// print processing
void print_proc(wchar_t buffer[BUFFER_SIZE], size_t *i);

// int(input()) processing
void iinput_proc(wchar_t buffer[BUFFER_SIZE], wchar_t
var[WORD_SIZE], size_t *i);

// end line
void end_line(wchar_t ch);

// assignment processing
void asgn_proc(wchar_t buffer[BUFFER_SIZE], wchar_t
vars[WORDS_SIZE][WORDS_SIZE],
               wchar_t var[WORD_SIZE], _Bool new, wchar_t
word[WORD_SIZE],
               size_t *cntv, size_t *i);
```

Листинг 3.2 – файл conpyc.c

```
#include "conpyc.h"
#include "strfunc.h"

// for loop condition (making brackets) processing
void for_cond_proc(wchar_t buffer[BUFFER_SIZE], size_t *i) {
  fputws(L"for (int ", stdout); // start for loop

  wchar_t var[WORD_SIZE] = {L'\0'}; // local variable
  wcpy_wrd_buf(buffer, var, i); // find local variable

  winsw_out(var);
  fputws(L"=0; ", stdout);
  winsw_out(var);
  putwchar(L'<');

  // find number
  skip_before_wch(buffer, L'(', i);

  // skip L'('
  (*i)++;

  // put number
  put_before_wch(buffer, L')', i);

  // skip L')'
  (*i)++;

  // skip L':'
  (*i)++;
```

```
  // close brackets
  fputws(L"; ", stdout);
  winsw_out(var);
  fputws(L"++)", stdout);
  if (*(buffer+(*i)) == L' ') putwchar(L' ');
  else fputws(L"\n\t", stdout);

}

// while loop and if condition (making brackets) processing
void whif_cond_proc(wchar_t buffer[BUFFER_SIZE], size_t *i) {
  // start while/if
  fputws(L" (", stdout);

  // skip L' '
  (*i)++;

  // put condition
  put_before_wch(buffer, L':', i);

  // skip L':'
  (*i)++;

  // close brackets
  putwchar(L')');
  if (*(buffer+(*i)) == L' ') putwchar(L' ');
  else fputws(L"\n\t", stdout);
}

// print processing
void print_proc(wchar_t buffer[BUFFER_SIZE], size_t *i) {
```

```
  // skip L"print" part
  skip_before_wch(buffer, L'(', i);

  // skip L'('
  (*i)++;

  // start printf func
  fputws(L"printf(\"%", stdout);

  // if string
  if (*(buffer+(*i)) == L'"') putwchar(L's');
  else putwchar(L'd');
  fputws(L"\\n\", ", stdout);

  // end print
  put_before_wch(buffer, L')', i);
  (*i)++;
  fputws(L");\n\t", stdout);
}

// int(input()) processing
void iinput_proc(wchar_t buffer[BUFFER_SIZE], wchar_t
var[WORD_SIZE], size_t *i) {
  // just skip all L"int(input())"
  skip_before_wspace(buffer, i);

  // put scanf for int
  fputws(L"scanf(\"%d\", &", stdout);
  winsw_out(var);
  fputws(L");\n\t", stdout);
}
```

```
// assignment processing
void asgn_proc(wchar_t buffer[BUFFER_SIZE], wchar_t
vars[WORDS_SIZE][WORDS_SIZE],
               wchar_t var[WORD_SIZE], _Bool new, wchar_t
word[WORD_SIZE],
               size_t *cntv, size_t *i) {
  // word copy to var
  wcpy(word, var);

  // skip spaces after variable
  skip_before_wch(buffer, L'=', i);

  // skip L'='
  if (*(buffer+(*i)) == L'=') (*i)++;

  // if variable isn't initialized
  if (new) {
    add_word_a(vars, var, *cntv);
    fputws(L"int ", stdout);
    (*cntv)++;
  }

  // print var and L'='
  winsw_out(var);
  fputws(L" = ", stdout);

  // read new word after L'='
  wcpy_wrd_buf(buffer, word, i);

  // int(input()), not input()!
```

```
  if (cmpnw(word, L"int", 3)) {
    if (new) {
      fputws(L"0;\n\t", stdout);
    }
    iinput_proc(buffer, var, i);
  } else {
    winsw_out(word);
    put_before_wch(buffer, L'\n', i);
    fputws(L";\n\t", stdout);
  }
}

// end line
void end_line(wchar_t ch) {
  if (ch==L'\n') {
    putwchar(L';');
    putwchar(ch);
    putwchar(L'\t');
  } else putwchar(ch);
}
```

Листинг 4.1 – файл main.c

```
#include <locale.h>
#include "strfunc.h"
#include "conpyc.h"

int main() {
  setlocale(LC_ALL, "");

  wchar_t buffer[BUFFER_SIZE] = {L'\0'};
  wchar_t word[WORD_SIZE] = {L'\0'};
  wchar_t var[WORD_SIZE] = {L'\0'};
  wchar_t vars[WORDS_SIZE][WORD_SIZE] = {{L'\0'}};
  wint_t ch = 0; // character
  size_t cntv=0; // number of stored variables
  size_t wlen=0; // word len
  size_t buf_len=0; // actualy buf size

  // read all input to buffer
  for (; buf_len<BUFFER_SIZE-1; buf_len++) {
    ch = getwchar();
    if (ch == WEOF || ch == L'\0') break;
    *(buffer+buf_len)=ch;
  }
  *(buffer+buf_len)=L'\0';

  // start c-file
  fputws(L"#include <stdio.h>\n\nint main() {\n\t", stdout);

  for (size_t i=0; i<buf_len; i++) {
    // read buffer
    ch = *(buffer+i);
```

```
    if (ch==L'\0') break;

    // check end of word (or just space)
    if (wisspace(ch)) {
      putwchar(ch);
    } else {
      wlen = wcpy_wrd_buf(buffer, word, &i); // extract a word
from input and cpy it to word
      if (wlen > 0) {
        // for loop
        if (cmpnw(word, L"for", 3)) for_cond_proc(buffer, &i);
        // if, while loopo
        else if (cmpnw(word, L"while", 5) || cmpnw(word, L"if",
2)) {
          winsw_out(word);
          whif_cond_proc(buffer, &i);
        // print
        } else if (cmpnw(word, L"print", 5)) {
          print_proc(buffer, &i);
        // assignment
        } else if (i+2<buf_len && *(buffer+i+1)==L'=' &&
*(buffer+i+2)!=L'=') {
          if (find_word_a(vars, word))
            asgn_proc(buffer, vars, var, 0, word, &cntv, &i);
          else asgn_proc(buffer, vars, var, 1, word, &cntv, &i);
        } else {
          winsw_out(word);
          end_line(ch);
        }
      }
    }
```

```
  }
  fputws(L"return 0;\n}", stdout);

  return 0;
}
```

Описание алгоритма распознавания конструкций

Алгоритм работы конвертера построен на последовательном чтении и анализе символов входного кода. Программа сначала полностью считывает весь текст из стандартного ввода в широкосимвольный буфер фиксированного размера. Затем начинается основной этап трансляции, где каждый символ буфера анализируется для выявления ключевых конструкций языка Python. Для выделения отдельных слов используется функция wcpy_wrd_buf, которая извлекает из буфера последовательности символов, ограниченные пробелами или специальными знаками.

После получения слова программа проверяет его соответствие известным ключевым словам, таким как «for», «if», «while», «print», «int». При обнаружении совпадения вызывается соответствующий обработчик, который выполняет преобразование синтаксиса Python в эквивалентный синтаксис C. Особенностью алгоритма является отслеживание объявленных переменных через специальный массив vars. Это позволяет реализовать автоматическое определение необходимости объявления переменной: при первом присваивании переменной добавляется ключевое слово «int», а при последующих использованиях только операция присваивания.

Алгоритм также обрабатывает отступы и переводы строк, сохраняя структуру исходного кода. Для циклов for выполняется преобразование конструкции типа «`for i in range(N)`» в стандартный цикл C «`for(int i=0; i<N; i++)`». Условные операторы и циклы while преобразуются путём замены двоеточия на круглые скобки и добавления фигурных скобок для обозначения тела. Ввод и вывод данных транслируются с учётом форматирования: функция `print` становится `printf` с указанием типа выводимого значения, а `int(input())` превращается в `scanf` с соответствующим спецификатором формата.

Примеры работы

Хороший пример (работает корректно)

Ввод (Python):

```
n = int(input())
for i in range(n):
    x = i * 2
    if x > 5:
        print(x)
```

Вывод (C):

```
#include <stdio.h>

int main() {
     int n = 0;
     scanf("%d", &n);
     for (int i=0; i<n; i++)
        int x = i * 2;
        if (x > 5)
            printf("%d\n", x);
     return 0;
}
```

Почему работает:

Хороший пример использует только поддерживаемые конструкции: ввод `int(input())` превращается в `scanf()`, цикл `for i in range(n)` корректно конвертируется в C-форму `for(int i=0; i<n; i++)`, а присваивания и вывод `print()` преобразуются в синтаксис C без ошибок типов. Все операции целочисленные, что соответствует возможностям конвертера.

Плохой пример (работает некорректно)

Ввод (Python):

```
name = "Ivan"
age = 25
print(name, age)
```

Вывод (C):

```
int main() {
     int name = "Ivan";
     int age = 25;
     printf("%d\n", name, age);
     return 0;
}
```

Почему не работает:

Конвертер не поддерживает строковые переменные, так как он не умеет объявлять переменные типа char[] или работать со строковыми литералами как с данными. При обработке строки `name = "Ivan"` программа попытается создать целочисленную переменную `name`, что приведёт к несоответствию типов. Кроме того, конвертер не способен корректно преобразовывать вызов `print()` с несколькими аргументами, так как он не умеет формировать сложные строки формата для `printf()`, которые требуются для одновременного вывода строк и чисел.

Вывод

Разработка конвертера выявила фундаментальные различия между языками Python и C на синтаксическом и концептуальном уровнях. Python, как язык высокого уровня, отличается минималистичным синтаксисом, динамической типизацией и автоматическим управлением памятью, что делает его удобным для быстрой разработки. C, напротив, требует явного указания типов данных, ручного управления памятью и более подробного описания операций, что обеспечивает больший контроль над ресурсами, но увеличивает сложность написания кода.

Синтаксически языки различаются в организации блоков кода: Python использует отступы, а C фигурные скобки. Циклы в Python построены на итераторах, тогда как в C применяются счётчики с явной инициализацией, условием и инкрементом (декрементом). Система ввода-вывода в C требует указания форматов данных, в то время как Python предоставляет универсальные функции print и input. Эти различия отражают разные философии разработки: Python ориентирован на удобство и скорость написания кода, а C на эффективность и низкоуровневый контроль.

Реализация конвертера показала, что автоматический перевод между языками возможен лишь для ограниченного подмножества конструкций, так как языки основаны на различных парадигмах и моделях выполнения. Python скрывает многие детали реализации, тогда как C требует их явного описания. Это делает C более производительным, но менее гибким в разработке. Конвертер демонстрирует, как высокоуровневые абстракции Python могут быть выражены через низкоуровневые конструкции C, что углубляет понимание внутреннего устройства языков программирования.