Министерство цифрового развития, связи и
массовых коммуникаций Российской Федерации
СибГУТИ

Отчёт по информатике
«Реализация простого конвертера с подмножества Python на Си»

Выполнил студент: Родионов П.А.
группы: ИВ-522
Вариант 9
Проверил ассистент кафедры ВС: Шевченко М.В.

Новосибирск 2025 г.

2. Исходный код конвертера на языке C

В данном разделе представлен полный исходный код программы-конвертера, реализованной на языке C.
Программа использует только базовые средства языка, разрешённые условиями задания: посимвольный ввод с помощью `getwchar()` и вывод с помощью `putwchar()`.
Структуры, `scanf`, `printf` и другие неразрешённые конструкции не используются в логике анализа входного Python-кода.

```
 1 | #include <stdio.h>
 2 | #include <wchar.h>
 3 | #include <locale.h>
 4 |
 5 | #define MAXINPUTARRAY 1000
 6 | #define MAXKNOWN 30
 7 | #define MAXKNOWNSYMB 32
 8 | #define MAXROW 80
 9 | #define CMDLIST 4
10 |
11 | // --- GLOBAL DATA ---
12 | wchar_t G_knownint[MAXKNOWN][MAXKNOWNSYMB] = {0};
13 | int G_knownCount = 0;
14 |
15 | // --- FUNCTION DECLARATIONS ---
16 | int it_is_what(wchar_t ch);
17 | int are_strings_equal(const wchar_t *s1, const wchar_t *s2);
18 | int is_variable_known(const wchar_t *name);
19 | void put_wchar_string(const wchar_t *s);
20 | void put_char_string(const char *s);
21 |
22 |
23 | // --- MAIN TRANSLATOR LOGIC ---
24 | int main() {
25 |     setlocale(LC_ALL, "");
26 |
27 |     wchar_t massive[MAXINPUTARRAY] = {0};
28 |     int CellsInMainArray = 0;
29 |
30 |     { // GET INPUT
31 |         wchar_t ch;
```

```
        while ((ch = getwchar()) != WEOF) {
            if (CellsInMainArray >= MAXINPUTARRAY) { return 1001; }
            massive[CellsInMainArray++] = ch;
        }
    }

    { // PRE-SCAN: DECLARE ALL VARIABLES AS INT
        put_char_string("#include <stdio.h>\n");
        put_char_string("int main(){\n");

        int k = 0;
        while (k < CellsInMainArray) {
            if (it_is_what(massive[k]) == 0 || it_is_what(massive[k]) == 1) {
                wchar_t tempName[MAXKNOWNSYMB] = {0};
                int len = 0;
                int startK = k;

                { // EXTRACT NAME
                    while (startK < CellsInMainArray && (it_is_what(massive[startK]) >= 0)) {
                        if (len < MAXKNOWNSYMB - 1) tempName[len++] = massive[startK];
                        startK++;
                    }
                }

                { // CHECK KEYWORDS AND DECLARE
                    int isKeyword = 0;
                    if (are_strings_equal(tempName, L"if") || are_strings_equal(tempName,
L"while") ||
                        are_strings_equal(tempName, L"for") || are_strings_equal(tempName,
L"print") ||
                        are_strings_equal(tempName, L"in") || are_strings_equal(tempName,
L"range") ||
                        are_strings_equal(tempName, L"int") || are_strings_equal(tempName,
L"input")) is
    Keyword = 1;

                    if (!isKeyword && tempName[0] != L'\0') {
                        if (!is_variable_known(tempName) && G_knownCount < MAXKNOWN) {
                            for (int c = 0; c < len; c++) G_knownint[G_knownCount][c] =
tempName[c];
                            G_knownCount++;

                            // DECLARE VARIABLE
                            put_char_string("int ");
                            put_wchar_string(tempName);
                            put_char_string(" = 0;\n");
                        }
```

```
                }
            }
            k = startK;
        } else {
            k++;
        }
    }
    putwchar(L'\n');
}

// --- SECOND PASS: LEXICAL ANALYSIS AND TRANSLATION ---
int massiveCursor = 0;
wchar_t row[MAXROW] = {0};
int deathMark = 1;
int blockLevel = 0;

do {
    if (massiveCursor >= CellsInMainArray) break;

    int command = 0;
    int rowCursor = 0;
    int initialSpaceCount = 0;

    { // GET ROW AND INDENTATION
        for (int i = 0; i < MAXROW; i++) row[i] = 0;
        rowCursor = 0;

        while (massiveCursor < CellsInMainArray && rowCursor < MAXROW - 1) {
            wchar_t cur = massive[massiveCursor];
            if (cur == L'\n') { massiveCursor++; break; }
            if (cur == WEOF || massiveCursor == CellsInMainArray - 1) { deathMark = 0; }
            row[rowCursor++] = cur;
            massiveCursor++;
        }

        int i = 0;
        while (row[i] == L' ' || row[i] == L'\t') { initialSpaceCount++; i++; }

        // CLOSE BLOCK
        if (blockLevel == 1 && initialSpaceCount == 0 && rowCursor > 0) {
            put_char_string("}\n");
            blockLevel = 0;
        }
```

```
116 |
117 |             // PRINT INDENTATION
118 |             for (int j = 0; j < initialSpaceCount; j++) putwchar(L' ');
119 |
120 |             // TRIM ROW
121 |             if (initialSpaceCount > 0) {
122 |                 for (int j = 0; j < rowCursor - initialSpaceCount; j++) {
123 |                     row[j] = row[j + initialSpaceCount];
124 |                 }
125 |                 rowCursor -= initialSpaceCount;
126 |                 for (int j = rowCursor; j < MAXROW; j++) row[j] = 0;
127 |             }
128 |         }
129 |
130 |         { // HANDLE PYTHON COMMENTS (#)
131 |             int hashPos = -1;
132 |             for (int i = 0; i < rowCursor; i++) {
133 |                 if (row[i] == L'#') { hashPos = i; break; }
134 |             }
135 |             if (hashPos != -1) {
136 |                 put_char_string("//");
137 |                 for (int k = 0; k < rowCursor; k++) {
138 |                     if (row[k] == L'#') break;
139 |                     putwchar(row[k]);
140 |                 }
141 |                 for (int k = hashPos + 1; k < rowCursor; k++) putwchar(row[k]);
142 |                 putwchar(L'\n');
143 |                 continue;
144 |             }
145 |         }
146 |
147 |         // SKIP EMPTY LINES
148 |         if (rowCursor == 0) {
149 |             putwchar(L'\n');
150 |             continue;
151 |         }
152 |
153 |         { // DETERMINE COMMAND TYPE
154 |             wchar_t cmd_list[CMDLIST][10] = {L"if", L"while", L"for", L"print"};
155 |             for (int i = 0; i < CMDLIST; i++) {
156 |                 int lenCmd = 0;
157 |                 while (cmd_list[i][lenCmd] != L'\0') lenCmd++;
158 |
```

```
159 |               if (rowCursor >= lenCmd) {
160 |                   int match = 1;
161 |                   for (int k = 0; k < lenCmd; k++) {
162 |                       if (row[k] != cmd_list[i][k]) { match = 0; break; }
163 |                   }
164 |
165 |                   if (match) {
166 |                       int pos = lenCmd;
167 |                       while(row[pos] == L' ') pos++;
168 |
169 |                       if (i == 3) { // PRINT
170 |                           if (row[pos] == L'(') { command = 4; break; }
171 |                       } else { // IF, WHILE, FOR
172 |                           int colonPos = -1;
173 |                           for (int p = pos; p < rowCursor; p++) {
174 |                               if (row[p] == L':') { colonPos = p; break; }
175 |                           }
176 |                           if (colonPos != -1) { command = i + 1; break; }
177 |                       }
178 |                   }
179 |               }
180 |           }
181 |       }
182 |
183 |       { // HANDLE ASSIGNMENT / OPERATION (command == 0)
184 |           if (command == 0) {
185 |               int eqPos = -1;
186 |               int plusEqPos = -1;
187 |               int varNameEnd = -1;
188 |
189 |               for (int i = 0; i < rowCursor; i++) {
190 |                   if (row[i] == L'=') {
191 |                       eqPos = i;
192 |                       if (i > 0 && row[i - 1] == L'+') plusEqPos = i;
193 |                       varNameEnd = i - 1;
194 |                       while(varNameEnd >= 0 && row[varNameEnd] == L' ') varNameEnd--;
195 |                       break;
196 |                   }
197 |               }
198 |
199 |               // If assignment found and not comparison
200 |               if (eqPos > 0 && row[eqPos + 1] != L'=' && plusEqPos == -1) {
201 |
```

```
202 |                 // 1. Check for input pattern
203 |                 wchar_t inputPattern[] = L"int(input())";
204 |                 int isInput = 1;
205 |                 int checkIdx = 0;
206 |                 int valPos = eqPos + 1;
207 |                 while (row[valPos] == L' ') valPos++;
208 |
209 |                 while (inputPattern[checkIdx] != L'\0') {
210 |                     if (row[valPos + checkIdx] != inputPattern[checkIdx]) { isInput = 0; break; }
211 |                     checkIdx++;
212 |                 }
213 |
214 |                 if (isInput) {
215 |                     // INPUT (scanf)
216 |                     put_char_string("scanf(\"%d\", &");
217 |                     for (int j = 0; j <= varNameEnd; j++) putwchar(row[j]);
218 |                     put_char_string(");\n");
219 |                 } else {
220 |                     // REGULAR ASSIGNMENT (x = -5, x = a + 1)
221 |                     for (int i = 0; i < rowCursor; i++) { putwchar(row[i]); }
222 |                     put_char_string(";\n");
223 |                 }
224 |                 continue;
225 |             }
226 |
227 |             // Augmented Assignment (x += 1) or other unknown lines
228 |             for (int i = 0; i < rowCursor; i++) { putwchar(row[i]); }
229 |             put_char_string(";\n");
230 |             continue;
231 |         }
232 |     }
233 |
234 |     { // TRANSLATE COMMAND (switch)
235 |         switch (command) {
236 |             case 1: // IF
237 |             case 2: // WHILE
238 |             case 3: { // FOR
239 |                 wchar_t *start_word;
240 |                 int len_cmd;
241 |                 if (command == 1) { start_word = L"if("; len_cmd = 2; }
242 |                 else if (command == 2) { start_word = L"while("; len_cmd = 5; }
243 |                 else { start_word = L"for("; len_cmd = 3; }
244 |
```

```
                put_wchar_string(start_word);
                int pos = len_cmd;
                while (row[pos] == L' ') pos++;

                if (command == 3) { // FOR (i in range(10))
                    wchar_t iterVar[MAXKNOWNSYMB] = {0};
                    int ivLen = 0;

                    // GET ITERATOR NAME
                    while (row[pos] != L' ' && row[pos] != L'\0' && pos < rowCursor &&
row[pos] != L
    ':') {
                        iterVar[ivLen++] = row[pos];
                        pos++;
                    }
                    // SKIP 'in range'
                    while (pos < rowCursor && row[pos] != L'(') pos++;
                    pos++;

                    // C FORMAT: for(i = 0; i < limit; i++)
                    put_wchar_string(iterVar); put_char_string(" = 0;");
                    put_wchar_string(iterVar); put_char_string(" < ");

                    // GET LIMIT
                    while (pos < rowCursor && row[pos] != L')' && row[pos] != L':') {
putwchar(row[p
    os++]); }

                    put_char_string(";");
                    put_wchar_string(iterVar); put_char_string("++");

                } else {
                    // IF / WHILE CONDITION (Handling x >= y conditions)
                    while (pos < rowCursor && row[pos] != L':') {
                        if (row[pos] == L'>' && row[pos+1] == L'=') {
                            put_char_string(" >= ");
                            pos += 2;
                        } else if (row[pos] == L'<' && row[pos+1] == L'=') {
                            put_char_string(" <= ");
                            pos += 2;
                        } else if (row[pos] == L'!' && row[pos+1] == L'=') {
                            put_char_string(" != ");
                            pos += 2;
                        } else if (row[pos] == L'=' && row[pos+1] == L'=') {
                            put_char_string(" == ");
```

```
                        pos += 2;
                    } else {
                        putwchar(row[pos++]);
                    }
                }
            }

            put_char_string(")\n");

            // OPEN BLOCK
            put_char_string("{\n");
            blockLevel = 1;
            break;
        }

        case 4: { // PRINT (print(x) or print("hello"))
            put_char_string("printf(");
            int pos = 5;
            while (row[pos] != L'(') pos++;
            pos++;

            int isString = (row[pos] == L'\"');

            if (isString) {
                putwchar(L'\"');
                pos++;
                while (pos < rowCursor && row[pos] != L')') {
                    if (row[pos] == L'\"' && isString) break;
                    putwchar(row[pos++]);
                }
                putwchar(L'\"');
            } else {
                put_char_string("\"%d\", ");
                while (pos < rowCursor && row[pos] != L')') {
                    putwchar(row[pos++]);
                }
            }
            put_char_string(");\n");
            break;
        }
    }
}
} while (deathMark);
```

```
    { // FINAL BRACKETS
        if (blockLevel == 1) {
            put_char_string("}\n");
        }
        put_char_string("}\n");
    }

    return 0;
}


// --- AUXILIARY FUNCTION DEFINITIONS ---
void put_wchar_string(const wchar_t *s) {
    int i = 0;
    while (s[i] != L'\0') { putwchar(s[i++]); }
}
void put_char_string(const char *s) {
    int i = 0;
    while (s[i] != '\0') { putwchar(s[i++]); }
}
int it_is_what(wchar_t ch) {
    if (ch >= L'a' && ch <= L'z') return 0;
    if (ch >= L'A' && ch <= L'Z') return 1;
    if (ch >= L'0' && ch <= L'9') return 2;
    if (ch == L'_') return 3;
    return -1;
}
int are_strings_equal(const wchar_t *s1, const wchar_t *s2) {
    int i = 0;
    while (s1[i] != L'\0' || s2[i] != L'\0') {
        if (s1[i] != s2[i]) return 0;
        i++;
    }
    return 1;
}
int is_variable_known(const wchar_t *name) {
    if (name[0] == L'\0') return 1;
    for (int r = 0; r < G_knownCount; r++) {
        if (are_strings_equal(G_knownint[r], name)) {
            return 1;
        }
    }
```

```
372 |     return 0;
373 | }
```

---

3. Описание алгоритма работы программы

Программа работает в несколько этапов.

Чтение входного текста

Весь входной Python-код считывается посимвольно из стандартного ввода с помощью `getwchar()` и сохраняется в массив фиксированного размера.

Предварительный анализ (первый проход)

На первом проходе программа просматривает весь входной текст и автоматически определяет все идентификаторы переменных.
Если идентификатор не является ключевым словом языка Python, он регистрируется как переменная и для него генерируется объявление типа `int` в начале результирующего C-кода.

Построчная обработка (второй проход)

Входной текст анализируется построчно с учётом отступов.
По отступам определяется начало и конец блоков условий и циклов.

Распознаваемые конструкции

Программа распознаёт следующие конструкции языка Python:

- оператор присваивания (`x = 5`, `x = a + 1`)
- ввод данных (`x = int(input())`)
- условный оператор `if`
- цикл `while`

- цикл `for ... in range(...)`
- оператор `print` для целых чисел и строк в двойных кавычках
- комментарии, начинающиеся с символа `#`

Генерация кода на C

Для каждой распознанной конструкции генерируется соответствующий фрагмент кода на языке C:

- Python-отступы преобразуются в фигурные скобки `{}`
- `print` преобразуется в `printf`
- `input()` преобразуется в `scanf`
- условия и циклы переводятся в синтаксис языка C

---

4. Примеры работы программы

Пример корректной работы

Вход (Python):

```
x = 5
y = -3
if x >= y:
    print(x)
```

Выход (C):

```
int x = 0;
int y = 0;

x = 5;
y = -3;
if(x >= y)
{
    printf("%d", x);
}
```

Данный пример успешно обрабатывается, так как используется допустимый поднабор языка Python: простые переменные, целые числа, условный оператор и `print`.

Пример некорректной обработки

Вход (Python):

```
x = [1, 2, 3]
print(x[0])
```

Причина ошибки

Данный код не будет корректно обработан программой, так как списки, индексация массивов и составные типы данных не входят в поддерживаемый поднабор языка Python. Конвертер предназначен только для работы с целыми числами и простыми выражениями.

---

5. Вывод

В ходе выполнения данной работы были изучены основные различия между языками Python и C на уровне синтаксиса и модели выполнения программ.

Python использует динамическую типизацию и отступы для задания структуры программы, в то время как язык C требует явного объявления типов и использует фигурные скобки для обозначения блоков кода.

Также было изучено, как высокоуровневые конструкции Python, такие как `input`, `print`, `if`, `while` и `for`, могут быть представлены в виде низкоуровневых конструкций языка C.
Работа позволила на практике понять принципы лексического анализа, трансляции кода и ограничения, возникающие при переводе между языками с разной моделью выполнения.