Лабораторна робота 3
З дисципліни “Системне програмування”
Студента групи МІ-31
Бідзілі Святослав Олексійовича

Постановка задачі:

Реалізувати лексичний аналізатор мови програмування. Для зберігання класів лексем організувати таблиці. Вивести вміст таблиць після обробки тексту програми.

Розрізняти принаймні такі класи лексем:

- числа (десяткові, з плаваючою крапкою, шістнадцяткові),
- рядкові та символьні константи,
- директиви препроцесора,
- коментарі,
- зарезервовані слова,
- оператори,
- розділові знаки,
- ідентифікатори.

Позначати ситуації з помилками (наприклад, нерозпізнавані символи).

Можливі варіанти виводу результату роботи програми:

- “розфарбування” тексту програми (наприклад, кольором),
- вивід пар < лексема , тип_лексеми >,
- вивести лексеми, що зустрілися в програмі, розбитими на класи.

1. Мова програмування Python.

Виконання

Для виконання я розробив програму на мові C++ . Весь код роботи можна знайти за посиланням:

https://github.com/anakib1/KNU_SP_2023_lab3

Пройдемося послідовно по коду:

С першу частина з створенням класу лексеми, заданням йому строкового представлення, кольору зафарбовки тексту та утиліти для виводу тексту в консоль. Тут немає жодного цікавого

елементу реалізації, лише заради зручності.

```
enum LexemType {
    NUM = 0,
    HEX,
    CNST,
    DIRECTIVE,
    KEYWORD,
    OPERATOR,
    PUNCTUATION,
    IDENTIFIER,
    SPACES,
    COMMENT,
    FUNCTION,
    UNKNOWN
};

string LexemTypeNames[] = { ... }

namespace Color { ... }
Color::Modifier LexemColors[] = { ... }

struct Lexem {
    LexemType type;
    string content;
    Lexem(LexemType type, string content) : type(type), content(content) {};
};
```

Більшість лексем повністю пояснюється своєю назвою. Функцією я називаю будь-який ідентифікатор, який «викликається». Коментарями вважаю лише однорядкові. Директив в мові пайтон немає. Константа – будь-яка рядкова константа.

Далі наступний фрагмент коду відповідає за перевірку належності рядка до якогось наперед заданого типу лексем. В принципі функції однотипні і перевіряють або очевидні речі, або належність рядка множині. Було використано set з стандартної бібліотеки.

```
set<string> punctiations;
set<string> operators;
set<string> operator_prefixes;
set<string> keywords;
set<string> identifier_starters;
set<string> identifier_suffixes;

bool is_space(string s) {
    if (s.size() != 1)return false;
    char c = s[0];
    return c == ' ' || c == '\n' || c == '\t' || c == '\r';
}

bool is_punct(string s) {
    return punctiations.count(s) > 0;
}
```

```
bool is_operator(string s) {
    return operators.count(s) > 0;
}
bool is_operator_prefix(string s) {
    return operator_prefixes.count(s) > 0;
}

bool is_character(string s) {
    return ('a' <= s[0] && s[0] <= 'z') || ('A' <= s[0] && s[0] <= 'Z');
}
bool is_digit(string s) {
    return '0' <= s[0] && s[0] <= '9';
}
bool is_keyword(string s) {
    return keywords.count(s) > 0;
}
bool is_identifier_start(string s) {
    return identifier_starters.count(s) > 0;
}
bool is_identifier_suffix(string s) {
    return identifier_suffixes.count(s) > 0;
}
```

Далі розглянемо основну функцію parse, яка приймає назву файла та повертає вектор лексем .

```
vector<Lexem> parse(const string filename) {
    ifstream f(filename);
    vector<Lexem> ret;
    string current = "";
    auto add = [&](LexemType type) {ret.push_back(Lexem(type, current)); current.clear(); };
    while (f.peek() != EOF){
        current.push_back(f.get());
        if (is_space(current)) add(LexemType::SPACES);
        else if (is_punct(current))
            add(LexemType::PUNCTUATION);
        else if (current == ".") {
            Lexem prev = ret.back();
            if (prev.type == LexemType::OPERATOR || prev.type == LexemType::IDENTIFIER || prev.type == LexemType::CNST) {
                add(LexemType::IDENTIFIER);
            }
            else {
                add(LexemType::UNKNOWN);
            }
        }
        else if (current == "\"" || current == "'") {
            while (!f.eof()) {
                char c = f.get();
                current.push_back(c);
                if (c == current[0]) break;
            }
            add(LexemType::CNST);
        }
```

Основна ідея – перевірити чи є новий символ початком якоїсь лексеми, дійти до кінця лексеми считуючи символи, додати її у відповідь. Це дуже легко реалізовується у випадку простих лексем – пропуска, пунктуації, тд.

Для перевірки того, що деякі ідентифікатори все ж є функціями, ми використовуємо трюк з тим, що переприсвоюємо їм значення, якщо наступна лексема виявляється скобкою (елемент

виклику).

```
else if (is_operator(current)) {
    while (!f.eof() && is_operator(string(1, f.peek()))) {
        current.push_back(f.peek());
        if (!is_operator(current) && !is_operator_prefix(current)) {
            current.pop_back();
            break;
        }
        else {
            f.get();
        }
    }
    if (!is_operator(current)) {
        add(LexemType::UNKNOWN);
    }
    else {
        if (current == "(") {
            if (ret.back().type == LexemType::IDENTIFIER)
                ret.back().type = LexemType::FUNCTION;
        }
        add(LexemType::OPERATOR);
    }
}
```

Окрема увага приділялася тому, щоб ідентифікатори правильно визначали точки. Можливі випадки виду arr.get_elemnt1().get2().state, які повинні правильно оброблятися.

```
else if (is_identifier_start(current)) {
    bool was_dot = false;
    while ( (was_dot == false && (is_identifier_suffix(string(1, f.peek())) || f.peek() == '.')) ||
        (was_dot == true && is_identifier_start(string(1, f.peek()))) {
        if (f.peek() == '.') was_dot = true;
        current.push_back(f.get());
    }
    if (is_keyword(current)) {
        add(LexemType::KEYWORD);
    }
    else {
        add(LexemType::IDENTIFIER);
    }
}
else if (is_digit(current)) {
    while (is_digit(string(1, f.peek())) || f.peek() == '.') {
        current.push_back(f.get());
    }
    if (is_digit(current)) add(LexemType::CNST);
    else add(LexemType::UNKNOWN);
}
else if (current == "#") {
    while (!f.eof() && f.peek() != '\n') {
        current.push_back(f.get());
    }
    add(LexemType::COMMENT);

}
else {
    cout << "WARN: SHOULD NOT HAPPEN!\n";
    add(LexemType::UNKNOWN);
```

Деякі лексеми задаються с конфігурації . Приклад конфігурації можна знайти в гітхабі. Шлях до неї вказується константою на початку програми.

```
void init() {
    ifstream f(CONFIG_PREFIX + "keywords.txt");
    string s;
    while (f >> s) {
        keywords.insert(s);
    }
    f.close();
    f.open(CONFIG_PREFIX + "operators.txt");
    while (f >> s) {
        operators.insert(s);
        for (int i = 1; i < s.size(); i++)
            operator_prefixes.insert(s.substr(0, i));
    }
    f.close();
    f.open(CONFIG_PREFIX + "punct.txt");
    while (f >> s) {
        punctiations.insert(s);
    }
    f.close();
    f.open(CONFIG_PREFIX + "identifier_start.txt");
    while (f >> s) {
        identifier_starters.insert(s);
    }
    f.close();
    f.open(CONFIG_PREFIX + "identifier_suffix.txt");
    while (f >> s) {
        identifier_suffixes.insert(s);
    }
    f.close();
}
```

Це дозволяє мінімальними змінами адаптувати код до нових кейвордів та інших змін в мові.

```
int main()
{
    init();
    cout << "Please input filename for parsing:\n>>> ";
    string filename; cin >> filename;
    auto res = parse(filename);
    for (auto x : res) {
        cout << x;
    }
}
```

В основній функції програма лише читає шлях до файлу і виводить результат – розфарбування за лексемами.

Приклади роботи:

Приклад 1

```
def f(x, y):
    if x > y:
        return y
    return x
```

Приклад 2

```
class AlternativeLightningModel(L.LightningModule):
    def __init__(self):
        super().__init__()
        self.loss = EncouragingLoss()
        self.model = nn.Sequential(nn.BatchNorm1d(21), nn.Linear(21, 30), nn.ReLU(), nn.Linear(30, 100), nn.ReLU(), nn.Linear(100, 1))
    def forward(self, X):
        return self.model(X)
    def training_step(self, batch, batch_id):
        X, y = batch
        X = X.view(X.size(0), -1)
        y_hat = F.sigmoid(self.model(X))
        y_hat = torch.stack([1 - y_hat, y_hat], dim = 1).squeeze()
        y_hat = torch.log(y_hat / (1 - y_hat))
        loss = self.loss(y_hat, y)
        self.log("train_loss", loss)
        return loss
    def validation_step(self, batch, batch_id):
        X, y = batch
        X = X.view(X.size(0), -1)
        y_hat = self.model(X)
        preds = F.sigmoid(y_hat)
        score = roc_auc_score(np.array(y.cpu()).reshape(-1), np.array(preds.cpu()).reshape(-1))
        self.log("val_auc", score, prog_bar=True, on_step=False, on_epoch=True)
        return score

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=1e-3)
        return optimizer

from sklearn.base import BaseEstimator, ClassifierMixin
class ModelWrapper(BaseEstimator, ClassifierMixin):
    def __init__(self, model):
        super().__init__()
        self.model = model
    def predict(self, X):
        return (self.predict_proba(X) > 0.5).astype(int)

    def predict_proba(self, X):
        with torch.no_grad():
            X = torch.tensor(np.array(X), dtype = torch.float)
            X = X.view(X.size(0), -1)
            y = self.model(X)
            if (y.shape[-1] == 2):
                y = F.softmax(y, dim = -1)[:, -1].cpu().numpy()
            else:
                y = F.sigmoid(y).cpu().squeeze().numpy()
            return y

    def fit(self, X, y):
        trainer = L.Trainer(max_epochs=100)
        X_dataset = data.TensorDataset(torch.tensor(X.to_numpy(), dtype = torch.float), torch.tensor(y.to_numpy()))
        trainer.fit(self.model, data.DataLoader(X_dataset, batch_size = 512, num_workers = 2))


class BoosterModelWrapper(BaseEstimator, ClassifierMixin):
    def __init__(self, params):
        super().__init__()
        self.model = None
        self.nb = params.pop('num_round')
        self.params = params
    def fit(self, X, y):
```

Приклад 3

```
#this program does something
def x(y, z):
        y += z # comment
```

Висновок

В цій лабораторній роботі ми змогли написати простий лексичний аналізатор для мови програмування пайтон.