UDC 004.42:811.111
DOI: https://doi.org/10.24919/2308-4634.2025.332796

Tetiana Cherniuk, *Senior Lecturer of the Foreign Languages Department, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine*
ORCID: https://orcid.org/0009-0005-1991-2303
Bohdan Somriakov, *Student of the Faculty of Computer Science, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine*
ORCID: https://orcid.org/0009-0003-7045-8586

THINKING LIKE A COMPILER: A SYSTEMATIC APPROACH TO SOLVING GRAMMATICAL TASKS IN EXAMS

Exams can be a source of stress for many students, particularly when it comes to solving grammatical tasks. One of the main difficulties arises from the natural tendency to approach these tasks in a disorganized way, attempting to juggle multiple mental processes at once. Students often find themselves simultaneously analyzing grammar rules, interpreting the meaning of sentences, considering the broader context, and even contemplating the emotional tone of the passage. This multitasking can be overwhelming, causing errors and incomplete answers as the pressure mounts.

However, there is an alternative way of thinking that could significantly improve a student's ability to handle such tasks: thinking like a computer. In the world of programming, computers process languages using a systematic approach, following precise rules to break down complex code into smaller, more manageable parts. Much like how a compiler processes programming languages – by separating syntax, semantics, and structure into clear steps – students can apply a similar strategy to language exams. By focusing on one aspect at a time and methodically working through the task, they can reduce mental overload and improve their accuracy.

This article draws an analogy between the way computers handle programming languages and how students can benefit from adopting a "compiler mindset" when tackling English exams. Through this lens, students will learn how to better organize their thought process, break down complex tasks, and avoid the common pitfalls that arise from trying to process too many elements at once. By approaching grammatical tasks in a structured, methodical way, students can increase both their confidence and performance under exam conditions.

This article researches the similarities between the functioning of compound languages and the structure of the English language and identifies how this knowledge can be applied to effective question answering in an English language exam. The findings allow for a better understanding of the patterns of language construction and their use in different contexts. This, in turn, contributes to the improvement of language competence and the successful passing of language tests.

Keywords: *grammar; English exams; NLP; BNF; language.*
Fig. 4. Tabl. 2. Ref. 15.

Тетяна Чернюк, *старший викладач кафедри іноземних мов Чорноморського національного університету імені Петра Могили, Миколаїв, Україна*
ORCID: https://orcid.org/0009-0005-1991-2303
Богдан Сомряков, *студент факультету комп'ютерних наук Чорноморського національного університету імені Петра Могили, Миколаїв, Україна*
ORCID: https://orcid.org/0009-0003-7045-8586

МИСЛИТИ ЯК КОМПІЛЯТОР: СИСТЕМНИЙ ПІДХІД ДО ВИРІШЕННЯ ГРАМАТИЧНИХ ЗАВДАНЬ НА ІСПИТАХ

Іспити можуть бути джерелом стресу для багатьох студентів, особливо коли йдеться про вирішення граматичних завдань. Основні труднощі виникають через природну тенденцію підходити до цих завдань неорганізовано, намагаючись жонглювати кількома розумовими процесами одночасно. Студенти часто водночас аналізують граматичні правила, інтерпретують значення речень, розглядають ширший контекст і навіть обмірковують емоційний тон уривку. Така багатозадачність може бути непосильною, спричиняючи помилки та неповні відповіді, оскільки тиск зростає.

Однак існує альтернативний спосіб мислення, який може значно покращити здатність учня справлятися з такими завданнями: мислення як у комп'ютера. У світі програмування комп'ютери обробляють мови, використовуючи системний підхід, дотримуючись точних правил, щоб розбити складний код на менші, більш керовані частини. Подібно до того, як компілятор обробляє мови програмування, розділяючи синтаксис, семантику та структуру на чіткі кроки, студенти можуть застосувати подібну стратегію до мовних іспитів. Зосереджуючись на одному аспекті за раз і методично опрацьовуючи завдання, вони можуть зменшити розумове перевантаження та підвищити свою точність.

У цій статті проводиться аналогія між тим, як комп'ютери працюють з мовами програмування, і тим, як студенти можуть отримати користь від прийняття "мислення компілятора" при підготовці до іспитів з *англійської мови. Під цим кутом зору студенти дізнаються, як краще організувати свій процес мислення, розбивати складні завдання на частини та уникати поширених помилок, які виникають при спробі обробити надто багато елементів одночасно. Підходячи до граматичних завдань у структурований, методичний спосіб, студенти можуть підвищити свою впевненість та покращити результати в умовах іспиту.*

Ключові слова: *граматика; іспити з англійської мови; NLP; BNF; мова.*

Introduction. When writing code, it's important to understand what happens behind the scenes during the compilation process. The compiler transforms the code into machine-readable instructions through three distinct stages, each playing a crucial role in this transformation:

Step 1: Lexical Analysis. The first step in the compilation process is lexical analysis. When you save your code in a. go file, the compiler's lexical analyzer breaks the code into individual tokens. These tokens are the building blocks of the programming language, including keywords (e.g., const), identifiers (e.g., a), literals (e.g., 9), and symbols (e.g., =).

Step 2: Syntax Analysis. The compiler's parser takes the tokens and checks if they form a valid program according to the language's syntax rules. This is where the compiler ensures that your code is structured correctly, with proper nesting, brackets, and semicolons.

Step 3: Semantic Analysis. After parsing, the compiler performs semantic analysis, which involves checking the meaning of the code. This is where the compiler checks the types of variables, ensures that operations are valid, and performs other semantic checks.

When students tackle English exam questions, they can benefit from a structured approach similar to the way compilers process code. Just as a compiler goes through three essential stages to transform code into machine-readable instructions, students can apply a three-step method to analyze and answer exam questions effectively.

Analysis of the latest relevant research and publications. Recent research and publications on the application of Backus-Naur Form (BNF) in English language processing and natural language processing (NLP), as well as the tools utilized in the development of English exams, underscore several key points:

- A Two-Stage BNF Specification of Natural Language. This paper presents a method of using BNF to specify natural language in such a way that a relatively small grammar of English can express the major grammatical constraints of the language and can be refined without undue proliferation of the rules. The results show that the departures of natural language from a context-free language are of a very restricted kind. The analysis obtained for sentences of the scientific literature is relevant for information processing [1].

- Backus-Naur form (BNF). The article provides an overview of Backus-Naur Form (BNF), a widely recognized meta-language used to describe the syntax of programming languages. It explains that BNF, named after its creators John W. Backus and Peter Naur, serves as a formal way to specify which sequences of symbols are considered syntactically valid within a programming language. The article emphasizes that while BNF effectively outlines the structural rules of a language, it does not address the semantics, or the meanings, of those valid sequences. It aims to discuss the fundamental concepts of BNF, examining how it functions as a tool for defining the syntax of programming languages and its importance in the realm of computer science [2].

- What is language? Part IV: BNF notation. Syntax diagrams. The article is part of a series that examines formal languages and their applications, specifically focusing on Backus-Naur Form (BNF) notation and its extensions. It addresses a gap in previous discussions about context-free grammars by providing practical examples of how BNF notation can be used to describe the syntax of languages [3].

- The restriction language for computer grammars of natural language. The paper introduces a programming language specifically designed to effectively and clearly express the restrictions applicable to natural language grammar. This language is built on ten years of experience from the N.Y.U. Linguistic String Project, which has focused on parsing English sentences. The language incorporates practical syntax and routines that facilitate computerized natural language analysis, and it is currently utilized in the implementation of the Linguistic String Parser [4].

- A Syntax-Based Analysis of Predication: Linguistic Structures. This article reviews a syntax-based analysis of predication in language, delving into its underlying linguistic structure. The research conducted employs analytical methods sourced from literature to comprehend sentence construction and the syntactic relationships forming predication [5].

- A Review of the Studies on the Frequent Administrations of English Tests. The aim of this paper is to give a review of the studies which have been conducted on the role of the frequent administrations of tests. This includes studies on the effect of testing frequency on students' scores, anxiety, motivation, preparation, class participation, long-term retention of the materials, and the effect of the feedback which is given based on students' performance on these frequent tests. It also gives a brief summary of different types of test-anxious students and models of test anxiety [6].

- The language of languages. The article discusses the foundational role of grammars in shaping various types of languages used in computing, including programming languages, query languages, and markup languages. It emphasizes that grammars determine the structure of these languages and introduces common notations for representing grammars, specifically Backus-Naur Form (BNF), Extended Backus-Naur Form (EBNF), and regular extensions to BNF [7].

The purpose of the research is to examine the similarities between the functioning of compiled languages and the structure of the English language, as well as to identify how these insights can be applied to effectively solve English exam questions.

Results of the research. BNF stands for Backus Naur Form notation. It is a formal method for describing the syntax of programming language which is understood as Backus Naur Formas introduced by John Bakus and Peter Naur in 1960.

For human consumption, a proper notation for encoding grammars intended and called Backus Naur Form (BNF). Different languages have different description and rules but the general structure of BNF is given below.

<name> ::= <expansion>

BNF (Backus-Naur Form) Rules Overview [8]:

1. Every name in Backus-Naur form is surrounded by angle brackets, < >, whether it appears on the left- or right-hand side of the rule.

2. An expansion is an expression containing terminal symbols and non-terminal symbols, joined together by sequencing and selection.

3. A terminal symbol may be a literal like (“$” or “function”) or a category of literals (like integer).

4. Simply juxtaposing expressions indicates sequencing.

A vertical bar | indicates choice.

Let’s examine the following table, which provides examples of mathematical expressions and indicates whether they are correct as seen in table 1.

№	Expression	Correctness
1	id$	Correct
2	id+id+id$	Correct
3	id+$	Wrong
4	id+(id+id)$	Correct
5	id+(id+id$	Wrong
6	id*(id+id)$	Correct
7	id(id+id$	Wrong

Table 1 – Examples of expressions and their correctness

The BNF for the mathematical expressions displayed in table 1 is structured as shown below:

<Expression> ::= <Term> | <Expression> "+" <Term>
<Term> ::= <Factor> | <Term> "*" <Factor>
<Factor> ::= "id" | "(" <Expression> ")"

Using the BNF, we can construct a table that serves as a foundation for the compiler’s parsing process as seen in table 2.

Non-terminal	Input Symbol					
	id	**+**	*****	**(**	**)**	**$**
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \textbf{id}$			$F \rightarrow (E)$		

Table 2 – Parsing table based on the BNF grammar

Table 2 is a parsing table derived from the BNF grammar to assist in parsing input strings. It consists of:

1. Non-terminals: Rows for non-terminal symbols (E, E', T, T', F).

2. Input symbols: Columns for terminal symbols (id, +, *, (,), and $).

3. Production rules: Cell values indicate applicable production rules for each non-terminal and input symbol.

For instance:

If the input symbol is id and the parser expects E, the rule E → TE' is applied.

If the input is + and the parser expects E', the rule E' → +TE' is used.

This table enables the parser to determine which production rules to apply, facilitating the construction of a predictive parser.

Figure 1 illustrates how a compiler can utilize the table to determine whether a mathematical expression is correctly constructed.

```
$ E id * ( id + id ) $
$ E' T id * ( id + id ) $
$ E' T' F id * ( id + id ) $
$ E' T' F id id * ( id + id ) $
$ E' T' * ( id + id ) $
$ E' T' F * * ( id + id ) $
$ E' T' F ( id + id ) $
$ E' T' ) E ( ( id + id ) $
$ E' T' ) E  id + id ) $
$ E' T' ) E' T  id + id ) $
$ E' T' ) E' T' F id + id ) $
$ E' T' ) E' T' id id + id ) $
$ E' T' ) E' T' + id ) $
$ E' T' ) E' + id ) $
$ E' T' ) E' T  + + id ) $
$ E' T' ) E' T  id ) $
$ E' T' ) E' T' F id ) $
$ E' T' ) E' T' id id ) $
$ E' T' ) E' T' ) $
$ E' T' ) E' ) $
$ E' T' ) ) $
$ E' T' $
$ E' $
$ $
CORRECT
```

Figure 1 – Compiler Usage of Parsing Table after inputting id * (id + id) $

Figure 2 illustrates how a compiler can utilize the table to determine whether a mathematical expression is wrongly constructed.

```
KeyError: '+'
$ E id * + id
$ E' T id * + id
$ E' T' F id * + id
$ E' T' id id * + id
$ E' T' * + id
$ E' T' F * * + id
$ E' T' F + id
```

Figure 2 – Compiler Usage of Parsing Table after inputting id * + id $

After analyzing simple mathematical examples, we can apply the same principles to more complex scenarios, extending their use to natural language processing, particularly English.

Let's examine a simplified representation of the English language using Backus-Naur Form (BNF). For example, the sentence "The cat sat on the mat." can be expressed in BNF as follows:

```
<sentence> ::= <noun_phrase> <verb_phrase>
<noun_phrase> ::= <determiner> <noun>
<verb_phrase> ::= <verb> <prepositional_phrase>
<prepositional_phrase> ::= <preposition> <noun_phrase>
<determiner> ::= "the"
<noun> ::= "cat" | "mat"
<verb> ::= "sat"
<preposition> ::= "on"
```

First, the sentence would be divided into tokens: ["The", "cat", "sat", "on", "the", "mat"].

Then the sentence would be divided into <noun_phrase> and <verb_phrase>:

["The", "cat", "sat", "on", "the", "mat"].

Then the <noun_phrase> would be divided into <determiner> <noun> and <verb_phrase> would be divided into <verb> <prepositional_phrase>:

["The", "cat", "sat", "on", "the", "mat"].

Then the <prepositional_phrase> would be divided into <preposition> <noun_phrase>:

["The", "cat", "sat", "on", "the", "mat"].

Then the <noun_phrase> would be divided into <determiner> <noun>:

["The", "cat", "sat", "on", "the", "mat"].

Then we convert everything into names as we cannot divide them any further:

[<determiner>, <noun>, <verb>, <preposition>, <determiner>, <noun>].

The tokens above do indeed follow the BNF structure which means that the sentence is correct.

Now, let's apply our knowledge with an actual example from a typical English test, as illustrated in Figure 3.

A. Choose the correct option

1.

- A. I think the show is about start now
- B. I the show is about to start now
- C. I think the show is about to start now

Figure 3 – Easy example from an actual English test [9]

The BNF of the exam sentence can be expressed as follows:

```
<sentence> ::= <subject> <verb> <object> <complement> <adverb>
<subject> ::= "I"
<verb> ::= "think"
<object> ::= <noun_phrase>
<complement> ::= <copula> <verb_phrase>
<copula> ::= "is"
<adverb> ::= "now"
<noun_phrase> ::= <determiner> <noun>
<determiner> ::= "the"
<noun> ::= "show"
<verb_phrase> ::= <preposition> <to> <verb>
<preposition> ::= "about"
<to> ::= "to"
<verb> ::= "start"
```

Remember that as sentences become longer, you don't need to create a complete BNF for each one as it will only waste your time during the exam. Instead, focus on areas where mistakes are likely to occur.

In sentence A we observe the phrase “about starting” which directly contradicts to the BNF of the English language in this context. In the BNF of the exam sentence we can clearly see that <verb_phrase> consists of 3 parts, one of which is <to>, which is clearly missing:

```
<sentence> ::= <subject> <verb> <object> <complement> <adverb>
<complement> ::= <copula> <verb_phrase>
<verb_phrase> ::= <preposition> <to> <verb>
<to> ::= "to"
```

This discovery clearly indicates that the absence of the <to> component within the <verb_phrase> renders the sentence grammatically incorrect, highlighting a critical flaw in its structure.

In sentence B we observe the phrase “I the show” which also directly contradicts to the BNF of the English language in this context. In the BNF of the exam sentence we can clearly see that <subject> is followed by <verb> and “the” is obviously not a verb:

```
<sentence> ::= <subject> <verb> <object> <complement> <adverb>
<subject> ::= "I"
<verb> ::= "think"
```

The lack of a suitable verb in the phrase leads to the conclusion that the entire sentence is incorrect and ungrammatical, underscoring a crucial flaw in its structural composition.

In sentence C, we find no grammatical errors or structural inconsistencies. Upon careful analysis, the sentence adheres to the syntactical rules as outlined in the BNF:

```
<sentence> ::= <subject> <verb> <object> <complement> <adverb>
<sentence> ::= "I" "think" "the" "show" "is" "about" "to" "start"
```

It may seem challenging and impractical to apply this approach during actual exams, but as sentences become more complex, being able to set aside their meanings and concentrate solely on syntax – much like compilers do – can significantly enhance your performance.

In fact, any English grammar rule can be expressed using BNF notation. Figure 4 presents the rules for constructing conditional sentences.

	Condition	Result
Zero	**If + Present Simple** If you take the street on the right	**Present Simple** It’s quicker
First	**If + Present Simple** If I finish work early	**Will/won’t + V1** I’ll go to the shop
Second	**If + Past Simple** If I wasn’t sick	**Would/wouldn’t + V1** I’d go to the party
Third	**If + Past Perfect** If I’d left earlier	**Would/wouldn’t have + V3** I wouldn’t have been late

Figure 4 – Conditional sentences rules [10]

The following code represents the conditional sentence rules from Figure 4, but in BNF notation:

```
<conditional_sentence> ::= <zero_conditional> | <first_conditional>
| <second_conditional> | <third_conditional>

<zero_conditional> ::= "if" <present_simple> "," <present_simple>
<first_conditional> ::= "if" <present_simple> "," <will_clause>
<second_conditional> ::= "if" <past_simple> "," <would_clause>
<third_conditional> ::= "if" <past_perfect> "," <would_have_clause>

<present_simple> ::= <subject> <verb_present>
<past_simple> ::= <subject> <verb_past>
<past_perfect> ::= <subject> "had" <verb_past_participle>
<will_clause> ::= <subject> "will" <verb_base_form>
<would_clause> ::= <subject> "would" <verb_base_form>
<would_have_clause> ::= <subject> "would have"
<verb_past_participle>
<subject> ::= "I" | "you" | "he" | "she" | "it" | "we" | "they"
<verb_present> ::= "do" | "is" | "has"
<verb_past> ::= "did" | "was" | "had"
<verb_past_participle> ::= "done" | "been"
<verb_base_form> ::= "do" | "be"
```

Only after performing syntax analysis and eliminating numerous incorrect options can you proceed to semantic analysis. This step allows you to understand the underlying logic [11] and determine why a specific answer is correct, particularly when syntax analysis alone is insufficient.

In summary, analyzing sentences in an exam context involves three essential stages: lexical analysis, syntax analysis, and semantic analysis.

1. Lexical analysis begins with identifying individual words and their classifications to ensure they conform to expected patterns [12].

2. Syntax analysis follows, where the structure of the sentence is examined to verify grammatical correctness and adherence to language rules [13].

3. Semantic analysis concludes the process by assessing the meaning and context, confirming that the sentence logically conveys the intended message [14].

By systematically applying these analysis stages, you can effectively evaluate sentence correctness, identify errors, and enhance your overall understanding of language in an exam setting [15].

Conclusion. Adopting a mindset akin to that of a compiler can significantly enhance one's problem-solving abilities. By approaching tasks stage by stage – beginning with lexical analysis, then syntax, and finally semantic analysis – individuals can minimize confusion and streamline their thought processes.

By following this systematic approach, students can clarify complex tasks, foster a deeper understanding of language, and ultimately improve their performance in English exams and beyond. Thinking like a compiler not only enhances comprehension but also equips individuals with valuable strategies for effective problem-solving.

This approach is especially beneficial for C2 grammar exams, where students may encounter unfamiliar words and even if they do not fully understand what the sentence means, they can still analyze syntax to derive the correct answer. Although this method may seem challenging at first, experience reveals that it becomes much easier over time, allowing students to navigate English exams with greater confidence and proficiency.

Moreover, this approach fosters a more comprehensive understanding of linguistic concepts, enabling students to connect theoretical principles with practical application. By thinking like a compiler, they gain insights into the systematic nature of language, empowering them to articulate their thoughts more clearly and effectively.

REFERENCES

1. A Two-Stage BNF Specification of Natural Language. Available at: https://www.tandfonline.com/doi/pdf/10.1080/01969727208542912 (Accessed 17 Feb. 2025).

2. Backus-Naur form (BNF). Available at: https://www.researchgate.net/publication/262254296_Backus-Naur_form_BNF (Accessed 17 Feb. 2025).

3. What is language? Part IV: BNF notation. Syntax diagrams. Available at: https://rafalhiszpanski.pl/en/2023/02/what-is-language-part-four/ (Accessed 17 Feb. 2025).

4. The restriction language for computer grammars of natural language. Available at: https://dl.acm.org/doi/10.1145/360881.360910 (Accessed 17 Feb. 2025).

5. A Syntax-Based Analysis of Predication: Linguistic Structures Available at: https://journal.aspirasi.or.id/index.php/Fonologi/article/download/195/220/879 (Accessed 17 Feb. 2025).

6. A Review of the Studies on the Frequent Administrations of English Tests. Available at: https://www.researchgate.net/publication/276248084_A_Review_of_the_Studies_on_the_Frequent_Administrations_of_English_Tests (Accessed 17 Feb. 2025).

7. The language of languages. Available at: https://matt.might.net/articles/grammars-bnf-ebnf/ (Accessed 17 Feb. 2025).

8. BNF Notation in Compiler Design. Available at: https://www.geeksforgeeks.org/bnf-notation-in-compiler-design/ (Accessed 17 Feb. 2025).

9. Level Test Upper Advanced C2. Available at: https://www.englishtag.com/tests_with_answers/level_test_upper_advanced_C2.asp#google_vignette (Accessed 17 Feb. 2025).

10. GRAMMAR: Conditional Sentences. Available at: https://www.languageunlimited.org/conditionals/ (Accessed 18 Feb. 2025).

11. The Importance of Logic and Critical Thinking. Available at: https://www.wired.com/2011/03/the-importance-of-logic-critical-thinking/ (Accessed 16 Feb. 2025).

12. Introduction to Lexical Analysis: What it is and How it Works. Available at: https://medium.com/@mitchhuang777/introduction-to-lexical-analysis-what-it-is-and-how-it-works-b25c52113405 (Accessed 18 Feb. 2025).

13. Introduction to Syntax Analysis. Available at: https://www.cs.mtsu.edu/~zdong/3210/OldSlides/SyntaxAnalyzerIntroduction.pdf (Accessed 17 Feb. 2025).

14. What is semantics? Available at: https://www.lenovo.com/us/en/glossary/what-is-semantics/#:~:text=Syntax%20refers%20to%20the%20grammatical,they%20relate%20to%20each%20other. (Accessed 16 Feb. 2025).

15. Language Understanding and Knowledge of Meaning. Available at: https://www.researchgate.net/publication/47697391_Language_Understanding_and_Knowledge_of_Meaning (Accessed 16 Feb. 2025).

Стаття надійшла до редакції 21.02.2025

"Будь-яке навчання людини, є не що інше, як мистецтво сприяти прагненню природи до свого власного розвитку".

Йоганн Генріх Песталоцці
видатний швейцарський педагог-новатор