Parallel Processing

Project 2 Phase2

CYK Algorithm and Context-Free Grammar

Students:

Al-Anoud Al-Subaie
Raghad Al-Suhaibani
Ftoon Al-Rubayea

Table of Contents:

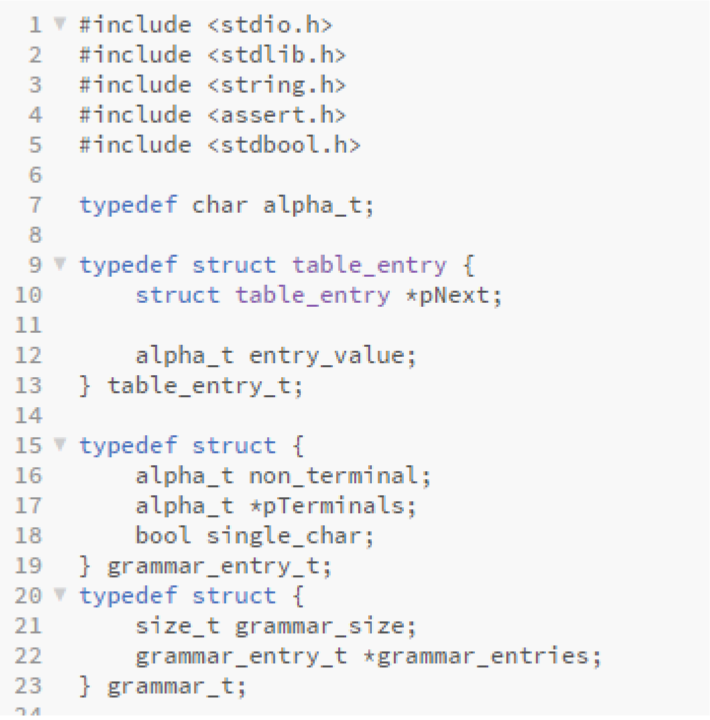
Table of Figure:

1.Problem Definition :

P1 . Write a multicore program that uses CYK algorithm [13] to parse a string of symbols. The inputs are a context-free grammar Gin Chomsky Normal Form and a string of symbols. At the end, the program should print YES if the string of symbols can be derived by the rules of the grammar and NO otherwise. Write a sequential program (no OpenMP directives at all) as well. Compare the running time of the sequential program with the running time of the multicore-program and compute the speedup for different grammars and different string lengths .

2.Write a Sequential Program Using CYK Algorithm :

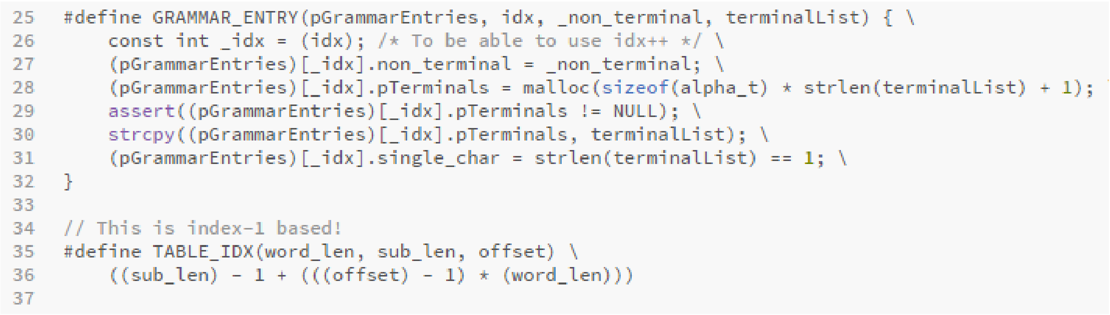
1.1 CONTEXT FREE GRAMMAR (CFG):



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <stdbool.h>

typedef char alpha_t;

typedef struct table_entry {
    struct table_entry *pNext;

    alpha_t entry_value;
} table_entry_t;

typedef struct {
    alpha_t non_terminal;
    alpha_t *pTerminals;
    bool single_char;
} grammar_entry_t;
typedef struct {
    size_t grammar_size;
    grammar_entry_t *grammar_entries;
} grammar_t;
```

Figure 0.1

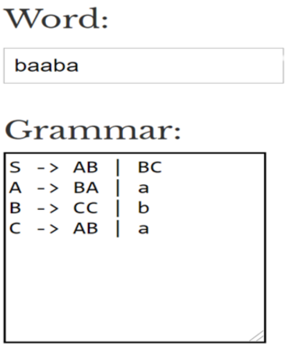
In **Figure** 0.1 initialise variable the table entry and the grammar entry .



```
#define GRAMMAR_ENTRY(pGrammarEntries, idx, _non_terminal, terminalList) { \
    const int _idx = (idx); /* To be able to use idx++ */ \
    (pGrammarEntries)[_idx].non_terminal = _non_terminal; \
    (pGrammarEntries)[_idx].pTerminals = malloc(sizeof(alpha_t) * strlen(terminalList) + 1);
    assert((pGrammarEntries)[_idx].pTerminals != NULL); \
    strcpy((pGrammarEntries)[_idx].pTerminals, terminalList); \
    (pGrammarEntries)[_idx].single_char = strlen(terminalList) == 1; \
}

// This is index-1 based!
#define TABLE_IDX(word_len, sub_len, offset) \
    ((sub_len) - 1 + (((offset) - 1) * (word_len)))

```

Figure 0.2

In **Figure** 0.2 this method will add non terminal and terminal in the grammar .



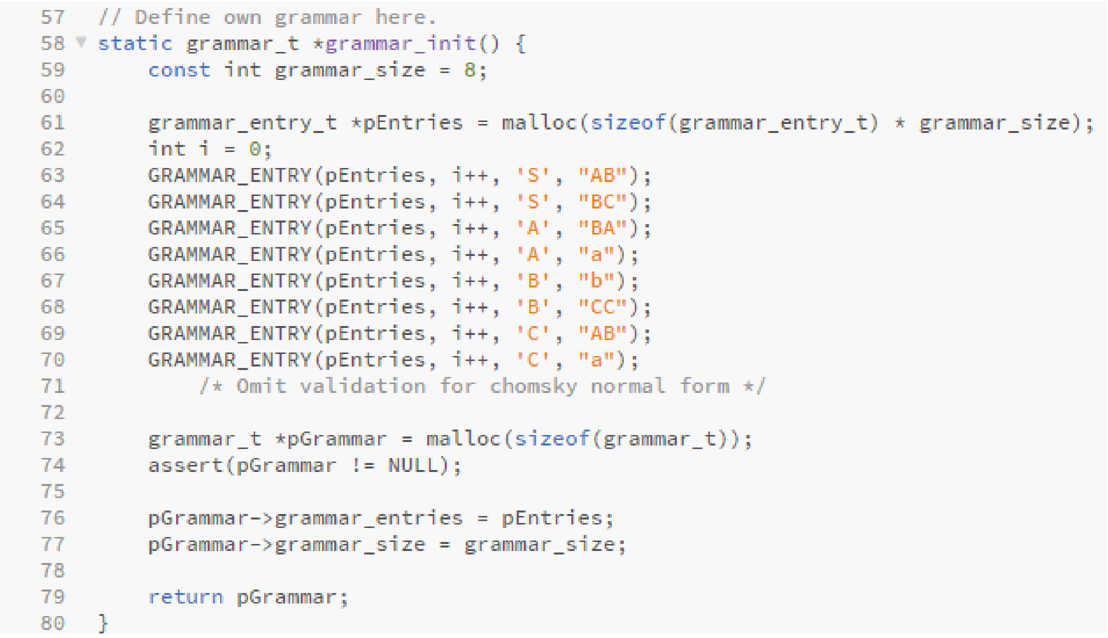
Word:

baaba

Grammar:

```
S  -> AB | BC
A  -> BA | a
B  -> CC | b
C  -> AB | a
```

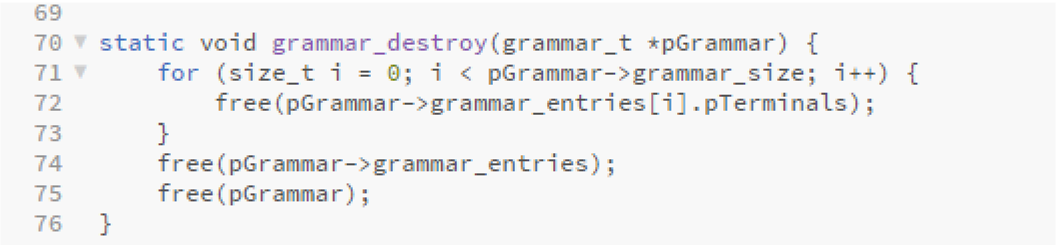
This is the grammar will add it to our code and testing using the word .



```
// Define own grammar here.
static grammar_t *grammar_init() {
    const int grammar_size = 8;

    grammar_entry_t *pEntries = malloc(sizeof(grammar_entry_t) * grammar_size);
    int i = 0;
    GRAMMAR_ENTRY(pEntries, i++, 'S', "AB");
    GRAMMAR_ENTRY(pEntries, i++, 'S', "BC");
    GRAMMAR_ENTRY(pEntries, i++, 'A', "BA");
    GRAMMAR_ENTRY(pEntries, i++, 'A', "a");
    GRAMMAR_ENTRY(pEntries, i++, 'B', "b");
    GRAMMAR_ENTRY(pEntries, i++, 'B', "CC");
    GRAMMAR_ENTRY(pEntries, i++, 'C', "AB");
    GRAMMAR_ENTRY(pEntries, i++, 'C', "a");
        /* Omit validation for chomsky normal form */

    grammar_t *pGrammar = malloc(sizeof(grammar_t));
    assert(pGrammar != NULL);

    pGrammar->grammar_entries = pEntries;
    pGrammar->grammar_size = grammar_size;

    return pGrammar;
}
```

Figure 0.3

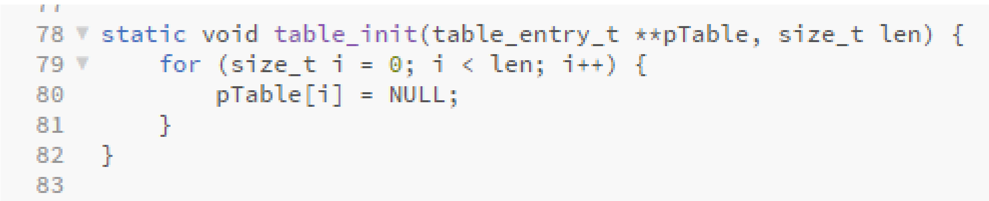
In **figure** 0.3 We will use the same grammer above, with grammer size of 8 and add it to methode GRAMMAR_ENTRY in figure 0.2 as table .



```c
static void grammar_destroy(grammar_t *pGrammar) {
    for (size_t i = 0; i < pGrammar->grammar_size; i++) {
        free(pGrammar->grammar_entries[i].pTerminals);
    }
    free(pGrammar->grammar_entries);
    free(pGrammar);
}
```

Figure 0.4

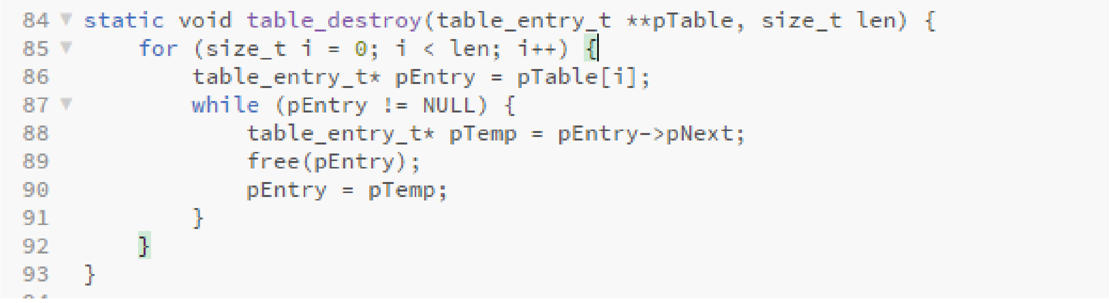
In **figure** 0.4 we will use this method when we are finished CYK algorithm because we are saving grammar in dynamic Memory Allocation “malloce” so we should use free for free from memory .



```c
static void table_init(table_entry_t **pTable, size_t len) {
    for (size_t i = 0; i < len; i++) {
        pTable[i] = NULL;
    }
}
```

Figure 0.5

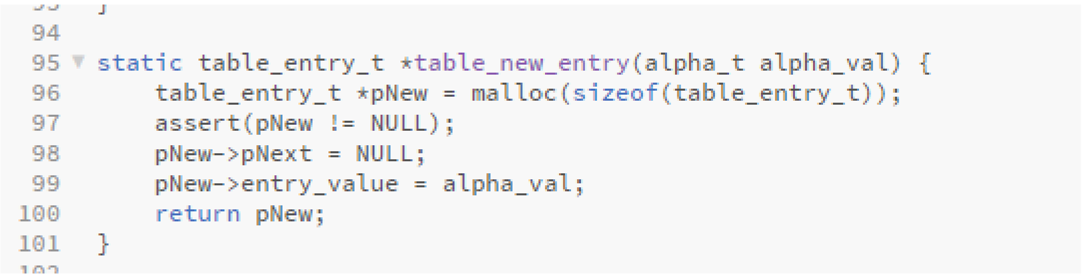
In **figure** 0.5 this method will initial table .



```c
static void table_destroy(table_entry_t **pTable, size_t len) {
    for (size_t i = 0; i < len; i++) {
        table_entry_t* pEntry = pTable[i];
        while (pEntry != NULL) {
            table_entry_t* pTemp = pEntry->pNext;
            free(pEntry);
            pEntry = pTemp;
        }
    }
}
```

Figure 0.6

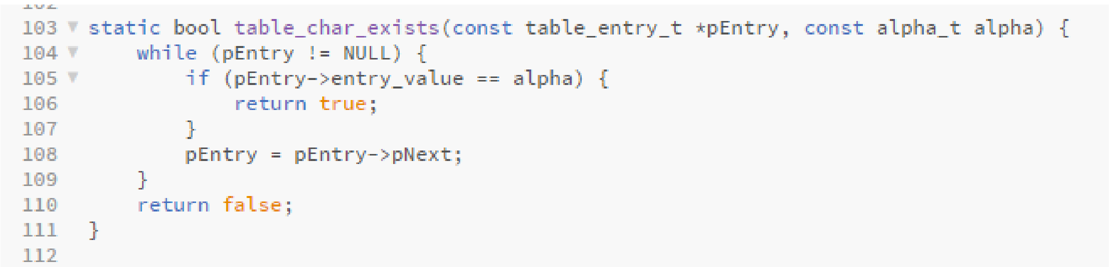
In **figure** 0.6 this method will when we are finished CYK algorithm because we are saving a table in dynamic Memory Allocation “malloce” so we should use free for free from memory .



```c
}

static table_entry_t *table_new_entry(alpha_t alpha_val) {
    table_entry_t *pNew = malloc(sizeof(table_entry_t));
    assert(pNew != NULL);
    pNew->pNext = NULL;
    pNew->entry_value = alpha_val;
    return pNew;
}
```

Figure 0.7

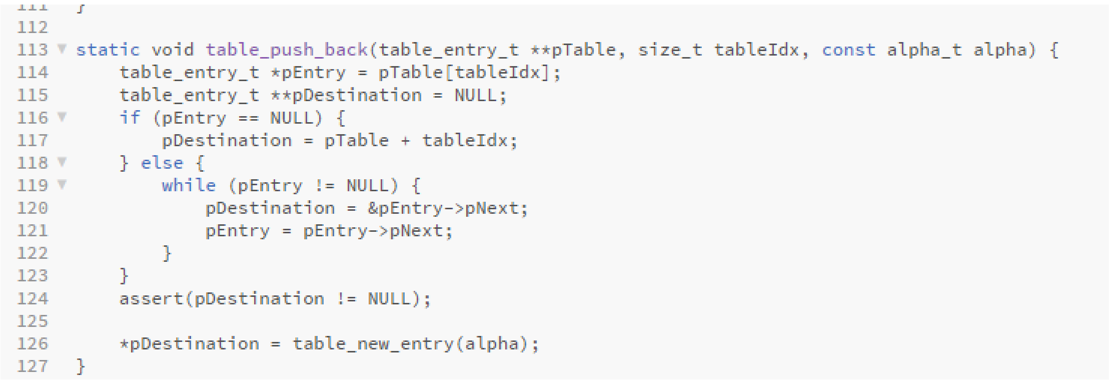
In **figure** 0.7 this method will use for entry table .



```c
static bool table_char_exists(const table_entry_t *pEntry, const alpha_t alpha) {
    while (pEntry != NULL) {
        if (pEntry->entry_value == alpha) {
            return true;
        }
        pEntry = pEntry->pNext;
    }
    return false;
}

```

Figure 0.8

In figure 0.8 this method will check if the char exists in the new table . if it is exists will return true ,otherwase false .

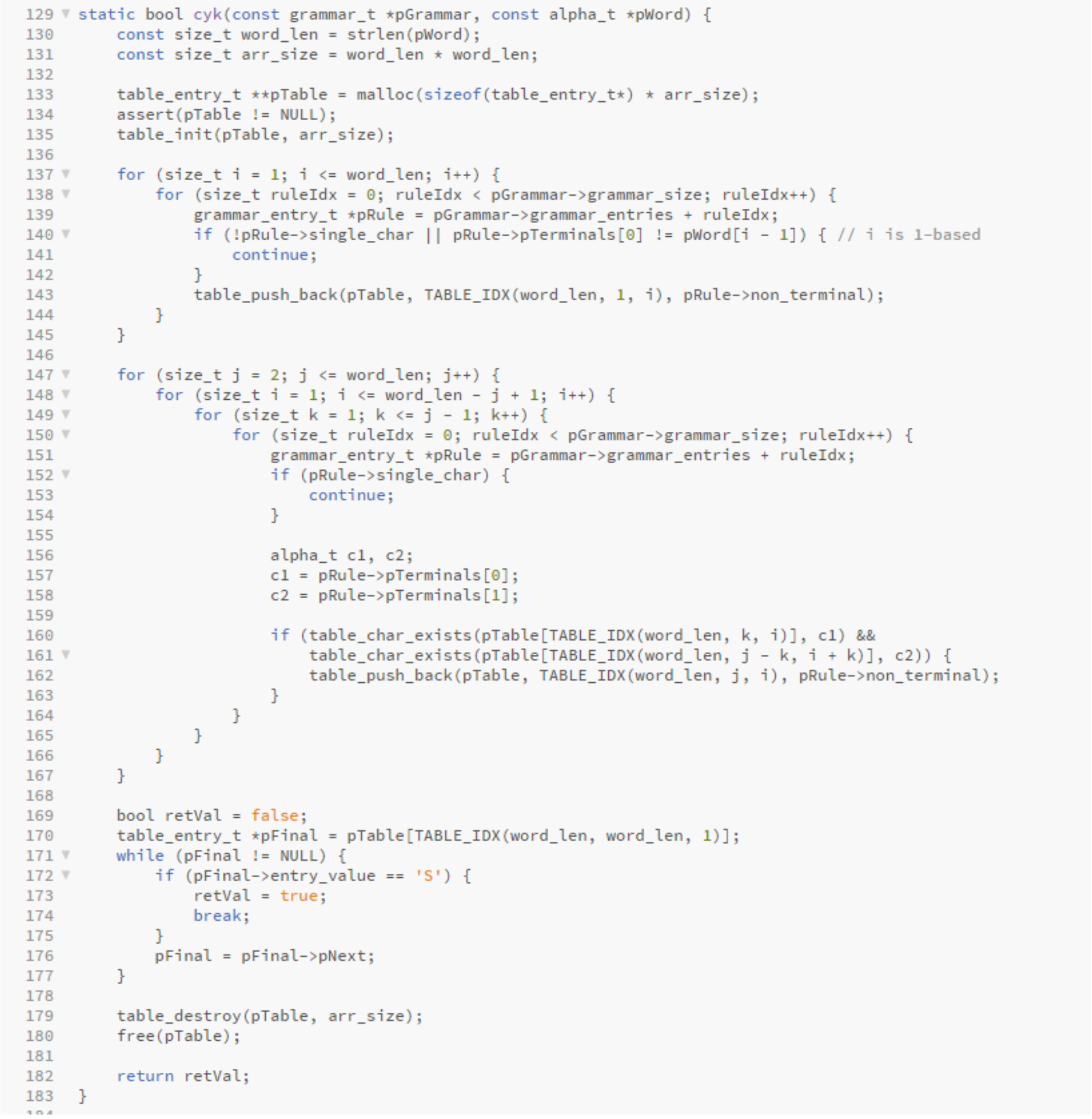


```c
}

static void table_push_back(table_entry_t **pTable, size_t tableIdx, const alpha_t alpha) {
    table_entry_t *pEntry = pTable[tableIdx];
    table_entry_t **pDestination = NULL;
    if (pEntry == NULL) {
        pDestination = pTable + tableIdx;
    } else {
        while (pEntry != NULL) {
            pDestination = &pEntry->pNext;
            pEntry = pEntry->pNext;
        }
    }
    assert(pDestination != NULL);

    *pDestination = table_new_entry(alpha);
}
```

Figure 0.9

In **figure** 0.9 this method will let table push back .

1.2 CYK Algorithm :

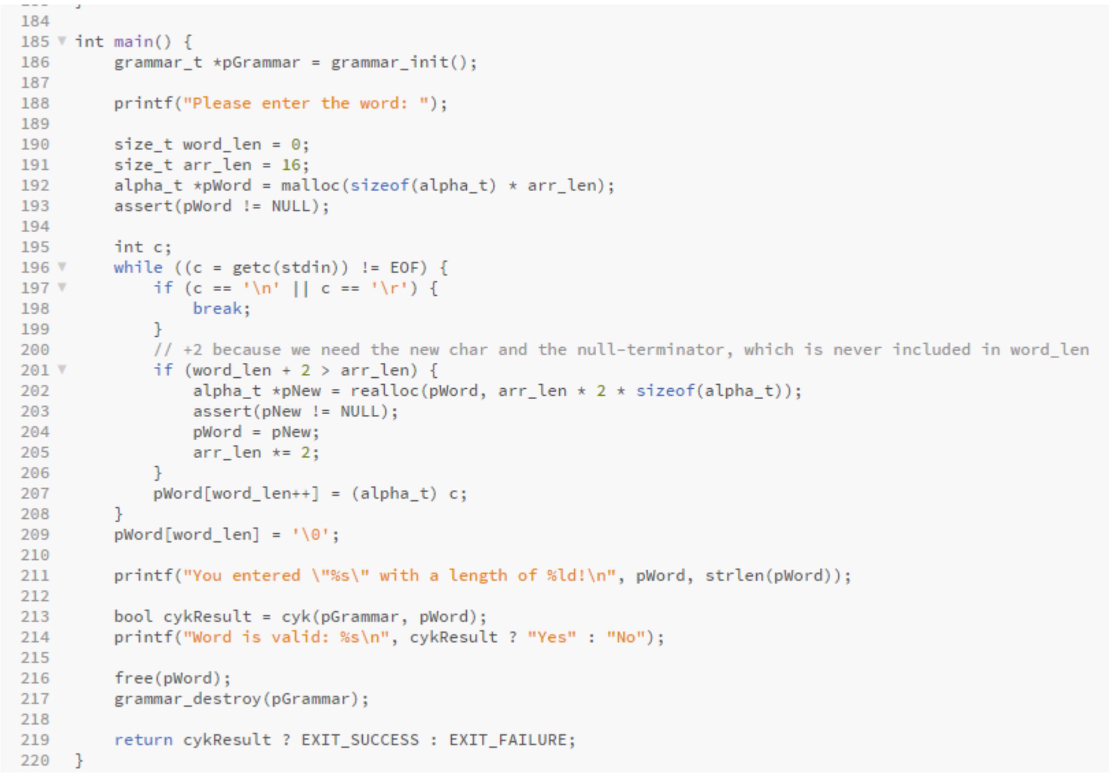


```c
static bool cyk(const grammar_t *pGrammar, const alpha_t *pWord) {
    const size_t word_len = strlen(pWord);
    const size_t arr_size = word_len * word_len;

    table_entry_t **pTable = malloc(sizeof(table_entry_t*) * arr_size);
    assert(pTable != NULL);
    table_init(pTable, arr_size);

    for (size_t i = 1; i <= word_len; i++) {
        for (size_t ruleIdx = 0; ruleIdx < pGrammar->grammar_size; ruleIdx++) {
            grammar_entry_t *pRule = pGrammar->grammar_entries + ruleIdx;
            if (!pRule->single_char || pRule->pTerminals[0] != pWord[i - 1]) { // i is 1-based
                continue;
            }
            table_push_back(pTable, TABLE_IDX(word_len, 1, i), pRule->non_terminal);
        }
    }

    for (size_t j = 2; j <= word_len; j++) {
        for (size_t i = 1; i <= word_len - j + 1; i++) {
            for (size_t k = 1; k <= j - 1; k++) {
                for (size_t ruleIdx = 0; ruleIdx < pGrammar->grammar_size; ruleIdx++) {
                    grammar_entry_t *pRule = pGrammar->grammar_entries + ruleIdx;
                    if (pRule->single_char) {
                        continue;
                    }

                    alpha_t c1, c2;
                    c1 = pRule->pTerminals[0];
                    c2 = pRule->pTerminals[1];

                    if (table_char_exists(pTable[TABLE_IDX(word_len, k, i)], c1) &&
                        table_char_exists(pTable[TABLE_IDX(word_len, j - k, i + k)], c2)) {
                        table_push_back(pTable, TABLE_IDX(word_len, j, i), pRule->non_terminal);
                    }
                }
            }
        }
    }

    bool retVal = false;
    table_entry_t *pFinal = pTable[TABLE_IDX(word_len, word_len, 1)];
    while (pFinal != NULL) {
        if (pFinal->entry_value == 'S') {
            retVal = true;
            break;
        }
        pFinal = pFinal->pNext;
    }

    table_destroy(pTable, arr_size);
    free(pTable);

    return retVal;
}
```

10.**0 Figure**

In **figure** 10.0 this cyk function will perform the cyk algorithm and return true when the entered string can be produced by the grammer, and otherwise it returns false.

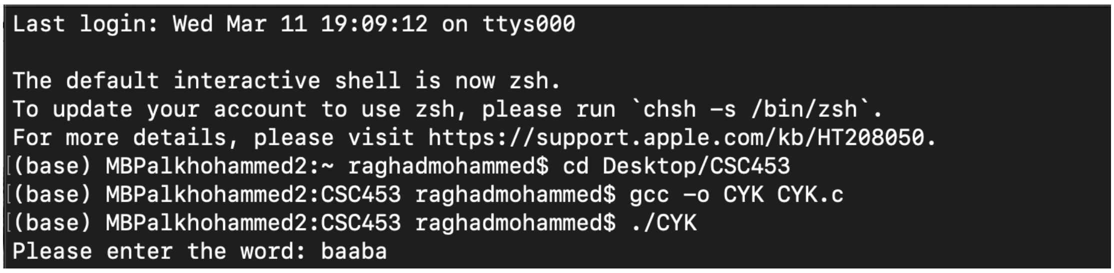
1.3 Main Method :



```
}

int main() {
    grammar_t *pGrammar = grammar_init();

    printf("Please enter the word: ");

    size_t word_len = 0;
    size_t arr_len = 16;
    alpha_t *pWord = malloc(sizeof(alpha_t) * arr_len);
    assert(pWord != NULL);

    int c;
    while ((c = getc(stdin)) != EOF) {
        if (c == '\n' || c == '\r') {
            break;
        }
        // +2 because we need the new char and the null-terminator, which is never included in word_len
        if (word_len + 2 > arr_len) {
            alpha_t *pNew = realloc(pWord, arr_len * 2 * sizeof(alpha_t));
            assert(pNew != NULL);
            pWord = pNew;
            arr_len *= 2;
        }
        pWord[word_len++] = (alpha_t) c;
    }
    pWord[word_len] = '\0';

    printf("You entered \"%s\" with a length of %ld!\n", pWord, strlen(pWord));

    bool cykResult = cyk(pGrammar, pWord);
    printf("Word is valid: %s\n", cykResult ? "Yes" : "No");

    free(pWord);
    grammar_destroy(pGrammar);

    return cykResult ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

Figure 0.11

In the main function **Figure** 0.11 , we can execute the CYK algorithm which will ask the user to enter the string to verify that belongs to the grammar and will display if the grammar belongs to yse or no .

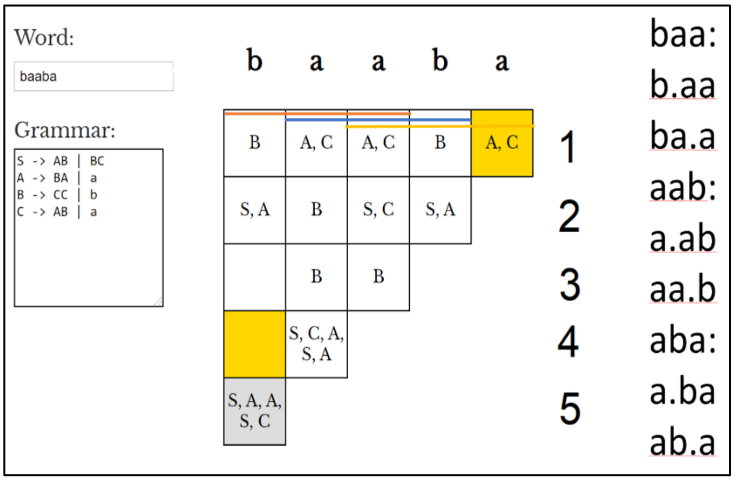


```
Last login: Wed Mar 11 19:09:12 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) MBPalkhohammed2:~ raghadmohammed$ cd Desktop/CSC453
(base) MBPalkhohammed2:CSC453 raghadmohammed$ gcc -o CYK CYK.c
(base) MBPalkhohammed2:CSC453 raghadmohammed$ ./CYK
Please enter the word: baaba
```

3.Results:

The program outputs if the entered string belong to the given grammar (Yes/No).

We are tasting this : it will getting yes .



Word:

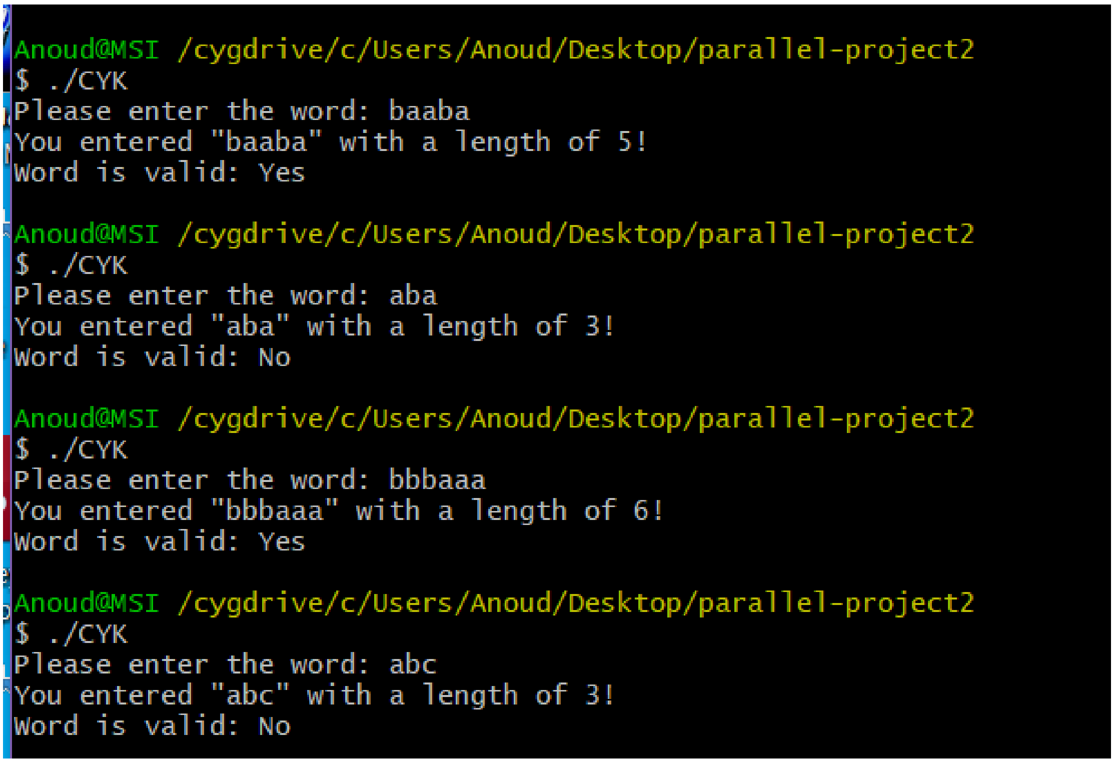
baaba

Grammar:

```
S -> AB | BC
A -> BA | a
B -> CC | b
C -> AB | a
```

b	a	a	b	a	
B	A, C	A, C	B	A, C	1
S, A	B	S, C	S, A		2
	B	B			3
	S, C, A, S, A				4
S, A, A, S, C					5

baa:
b.aa
ba.a
aab:
a.ab
aa.b
aba:
a.ba
ab.a



```
Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ ./CYK
Please enter the word: baaba
You entered "baaba" with a length of 5!
Word is valid: Yes

Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ ./CYK
Please enter the word: aba
You entered "aba" with a length of 3!
Word is valid: No

Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ ./CYK
Please enter the word: bbbaaa
You entered "bbbaaa" with a length of 6!
Word is valid: Yes

Anoud@MSI /cygdrive/c/Users/Anoud/Desktop/parallel-project2
$ ./CYK
Please enter the word: abc
You entered "abc" with a length of 3!
Word is valid: No
```

12.0 Figure