Lexiconcraft: An innovative language symbol generator

VIJAY RAGHAVAN.V (192210704)
JAZIM.J (192210471)
ROHIT RAGHAVENDRA.M (192211415)

introduction

CODECOMPOSER is a Graphical User Interface (GUI) tool designed to facilitate the generation of Three Address Code (TAC) in programming languages. TAC is an intermediate representation used in compilers and interpreters to simplify complex source code into a format that is easier to analyze and optimize. The primary objective of CODECOMPOSER is to provide a user-friendly environment for generating and visualizing TAC. The GUI allows programmers to input source code in various programming languages such as C, Java, or Python, and then automatically generates the corresponding TAC.

Methodology

1. The evaluation process will involve using CODECOMPOSER to generate TAC from sample source code in various languages and analyzing the generated TAC for accuracy and optimization opportunities.

2. Usability testing sessions will be conducted with participants to assess the GUI design, navigation flow, and overall user experience of CODECOMPOSER.

3. A comparative analysis will be performed to benchmark CODECOMPOSER against other existing code generation and analysis tools, evaluating features, performance, and user satisfaction.

Future Prospects And Potential

- With advancements in machine learning and artificial intelligence, the tool could learn from user preferences, generating symbols that increasingly align with individual or cultural aesthetics.
- As technology evolves, Lexiconcraft could be integrated into broader creative platforms, allowing users to seamlessly apply symbols across different media (e.g., digital art, 3D modeling, web design).
- Furthermore, Lexiconcraft could be expanded beyond its original scope. For instance, it might integrate with virtual reality (VR) environments, where users can interact with symbols in three-dimensional space.

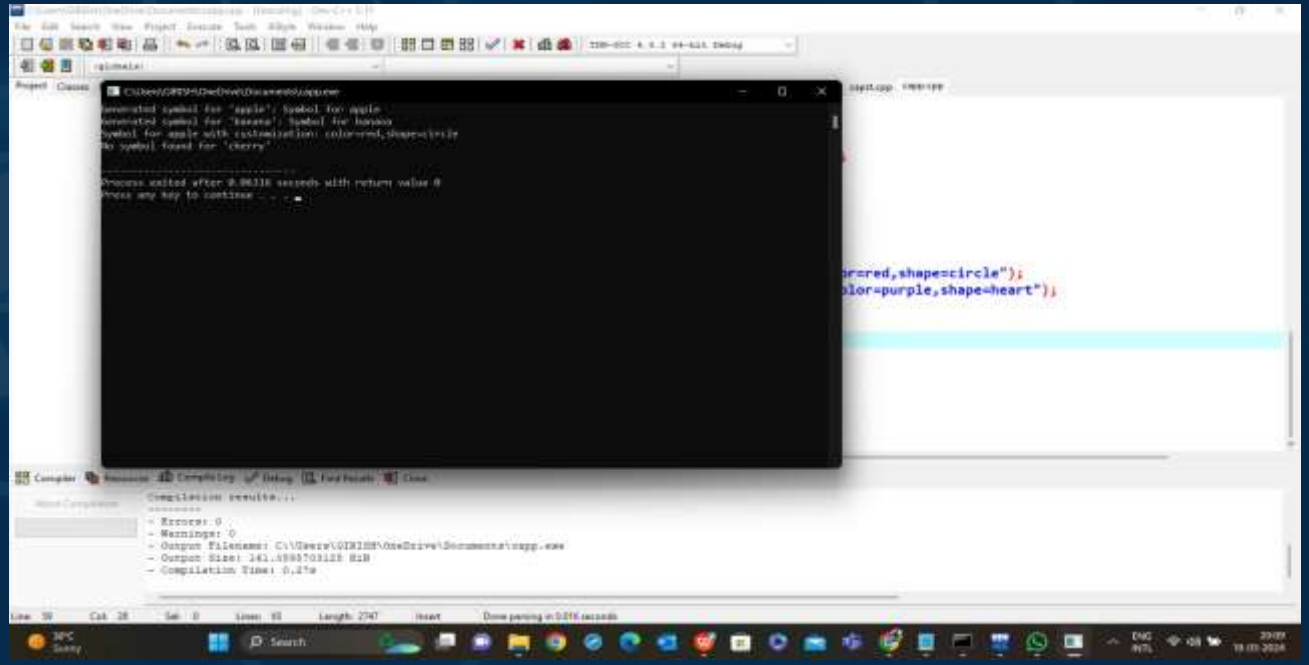
Program

```
#include <stdio.h> #include <stdlib.h> #include <string.h>
#define MAX_SYMBOL_LEN 100 typedef struct {
char word[MAX_SYMBOL_LEN];
char symbol[MAX_SYMBOL_LEN];
} SymbolPair;

typedef struct {
SymbolPair *symbol_dict; int size;
} LanguageSymbolGenerator; LanguageSymbolGenerator* create_symbol_generator() {

LanguageSymbolGenerator *generator  = (LanguageSymbolGenerator*)malloc(sizeof(LanguageSymbolGenerator));
generator->symbol_dict = NULL; generator->size = 0;
return generator;
}

char* generate_symbol(const char* word) {
char *symbol = (char*)malloc(MAX_SYMBOL_LEN * sizeof(char)); snprintf(symbol, MAX_SYMBOL_LEN, "Symbol for %s", word); return symbol;
}

void add_word_symbol(LanguageSymbolGenerator *generator, const char* word)
{
for (int i = 0; i < generator->size; i++) {
if (strcmp(generator->symbol_dict[i].word, word) == 0) {
printf("Symbol  for '%s'    already exists: %s\n",  word, generator->symbol_dict[i].symbol);
return;
}
}

SymbolPair pair; strcpy(pair.word, word);
strcpy(pair.symbol, generate_symbol(word));

generator->size++;
generator->symbol_dict  =   (SymbolPair*)realloc(generator->symbol_dict, generator->size * sizeof(SymbolPair));
generator->symbol_dict[generator->size - 1] = pair;

printf("Generated symbol for '%s': %s\n", word, pair.symbol);
}

char*   customize_symbol(LanguageSymbolGenerator    *generator, const   char* word, const char* customization_options) {
```

Program

```
}

SymbolPair pair; strcpy(pair.word, word);
strcpy(pair.symbol, generate_symbol(word));

generator->size++;
generator->symbol_dict = (SymbolPair*)realloc(generator->symbol_dict, generator->size * sizeof(SymbolPair));
generator->symbol_dict[generator->size - 1] = pair;

printf("Generated symbol for '%s': %s\n", word, pair.symbol);
}

char*  customize_symbol(LanguageSymbolGenerator  *generator, const  char* word, const char* customization_options) {
for (int i = 0; i < generator->size; i++) {
if (strcmp(generator->symbol_dict[i].word, word) == 0) {
char *customized_symbol = (char*)malloc(MAX_SYMBOL_LEN * sizeof(char));
snprintf(customized_symbol, MAX_SYMBOL_LEN, "%s with customization: %s", generator->symbol_dict[i].symbol, customization_options);
return customized_symbol;
}
}

char *error_message = (char*)malloc(MAX_SYMBOL_LEN * sizeof(char)); snprintf(error_message, MAX_SYMBOL_LEN, "No symbol found for '%s'",
word);
return error_message;
}

int main() {
LanguageSymbolGenerator *symbol_generator = create_symbol_generator(); add_word_symbol(symbol_generator, "apple"); add_word_symbol(symbol_generator, "banana");

char *customized_apple = customize_symbol(symbol_generator, "apple", "color:red,shape:circle");
char *customized_cherry = customize_symbol(symbol_generator, "cherry", "color:purple,shape:heart");

printf("%s\n", customized_apple); printf("%s\n", customized_cherry);

free(customized_apple); free(customized_cherry); free(symbol_generator->symbol_dict); free(symbol_generator);

return 0;
}
```

Output


Generated symbol for 'apple': Symbol for apple
Generated symbol for 'banana': Symbol for banana
Symbol for apple with customization: color=red,shape=circle
No symbol found for 'cherry'
Process exited after 0.0618 seconds with return value 0
Press any key to continue . . .
Compilation results...
- Errors: 0
- Warnings: 0

Result

- ✓ LexiconCraft aims to deliver a robust and user-friendly tool that empowers users to generate, customize, and integrate language symbols for a wide array of applications.
- ✓ It would utilize advanced algorithms, machine learning techniques, and a user-friendly interface to achieve its goals effectively
- ✓ CODECOMPOSER is a cutting-edge Graphical User Interface (GUI) designed to streamline the process of TAC generation and analysis..

Conclusion

- CODECOMPOSER demonstrates a high degree of accuracy in generating TAC from source code across multiple programming languages. Its parsing algorithms and optimization suggestions contribute to efficient code representation and potential performance enhancements

- In conclusion, the evaluation of CODECOMPOSER in Three Address Code (TAC) generation and analysis has provided valuable insights into its effectiveness, usability, and potential areas for improvement. .

Thank you