Relazione Homework 1: `myPreCompiler`

Francesco Zompanti 2012601
Vladut Florian Stanciu 2012601
Luca Panetta 2012601
Corso di Sistemi Operativi 2

Obiettivo e contesto

Il progetto `myPreCompiler` implementa un preprocessore testuale per file sorgente in linguaggio C, simulando parte del comportamento di un compilatore reale. L'obiettivo è analizzare e trasformare un codice sorgente secondo specifiche richieste, garantendo robustezza, modularità e efficienza.

Architettura del sistema

L'applicazione è progettata con un'architettura modulare, suddivisa nei seguenti componenti:

- `main.c`: parsing argomenti CLI, gestione controllo flusso e orchestrazione.
- `preprocessor.c`: logica di espansione `#include`, eliminazione commenti e analisi identificatori.
- `utils.c`: funzioni ausiliarie per gestione file, parsing riga per riga, logging, controllo sintattico.
- `myPreCompiler.h`: dichiarazioni centralizzate di strutture dati, funzioni e macro.

Ogni modulo utilizza esclusivamente interfacce definite nell'header, garantendo basso accoppiamento e alta coesione.

Funzionalità implementate

Il programma processa il file sorgente attraverso una pipeline testuale:

1. **Parsing degli argomenti:** riconoscimento e validazione delle opzioni `-in`, `-out`, `-verbose`.
2. **Espansione delle direttive `#include`:** inclusione ricorsiva del contenuto dei file, assunti presenti nella working directory.
3. **Rimozione dei commenti:** rimozione dei commenti inline (//) e multilinea (/* */) tramite regex, mantenendo intatte le righe del codice per non alterare la numerazione originale.
4. **Validazione identificatori:** controllo lessicale delle dichiarazioni locali e globali, con logging di identificatori non validi (es. contenenti caratteri illegali o inizianti con cifre).
5. **Generazione dell'output:** scrittura del codice trasformato su file o `stdout`, a seconda delle opzioni.
6. **Statistiche opzionali:** in modalità verbose, vengono tracciati: numero di righe eliminate, file inclusi, variabili controllate, errori individuati, dimensione e righe del file in input/output.

Strutture dati e gestione della memoria

Le funzioni di I/O testuale impiegano allocazione dinamica per gestire file di dimensione arbitraria. Le stringhe vengono lette riga per riga tramite buffer ridimensionabili, mentre le liste di errori, file inclusi e variabili analizzate vengono gestite tramite array dinamici.

La memoria viene sempre deallocata esplicitamente a fine elaborazione o in caso di errore fatale, secondo uno schema LIFO, garantendo l'assenza di memory leak.

Gestione errori e robustezza

Il sistema gestisce:

- Errori di sintassi nei parametri CLI.
- File non esistenti o non leggibili.
- Inclusioni non risolte o ricorsive.
- File corrotti o vuoti.
- Errori di scrittura su output.

In tutti i casi, l'output dell'errore è rediretto su `stderr` e l'uscita avviene con codice coerente (`EXIT_FAILURE`).

Test e validazione

La validazione è avvenuta attraverso i seguenti casi di test:

- `test_comments.c`: verifica corretta eliminazione commenti singoli e multilínea.
- `test_identifiers.c`: identificazione variabili con sintassi invalida.
- `test_include_main.c`, `test_include_header.h`: inclusione diretta e transitiva.
- `test_malformed_include.c`: gestione robusta di direttive include errate o mancanti.

L'output trasformato è stato confrontato con l'originale tramite `diff`, e le statistiche sono state ispezionate per coerenza.

Conclusioni

Il progetto `myPreCompiler` implementa un preprocessore funzionale, modulare e facilmente estendibile. Sono stati rispettati tutti i vincoli architetturali richiesti e validati tutti i casi di test.