COMP LabBook 2024 1 - E4

Lucas M. Schnorr

December 9, 2024

Contents

1 Introdução

- Arquivo `e4_make.log` contém o log de compilação
- Arquivo `e4.log` contém o log de execução dos testes
- Arquivo `e4_arquivo.log` contém também o log de execução dos testes, mas simplificado para transformação em formato tabular e contagem das respostas.
- Arquivo `e4_error_code.csv` contém o mapeamento do nome para o código de erro, de maneira a intepretar os nomes dos erros nos arquivos de testes
- Arquivo `e4_expected_code.csv` contém o código esperado para cada um dos testes, que pode ser o nome quando trata-se de um erro semântico, ou 0 quando não existe erro esperado.
- Arquivo `e4_objetivo.csv` fornece um sumário da nota objetiva.
  - A coluna **N** indica a quantidade de testes, a coluna **Correto** indica a quantidade de testes correto, e a coluna **E4.O** a nota objetiva final, que é oriunda de Correto/N.
- Arquivo `e4_tests_with_expected.csv` fornece a decisão individual de cada teste. A Coluna **Test** indica o teste utilizado (nome do arquivo), a coluna **Res** indica a resposta fornecida pelo programa do grupo (obtida a partir de $? após executar o programa em um script bash), a coluna **Output** indica o conteúdo da saída padrão fornecida pelo programa (quando existem múltiplas linhas, apenas a primeira é mostrada), a coluna **Code** indica o código de erro esperado para o teste, e, por fim, a coluna **TEST.WORKED** indica se o teste passou (TRUE) ou não (FALSE).

Para os testes que causam falha de segmentação, use o **gdb** para identificar o porquê após compilar seu programa com a opção -g para o compilador.

De uma maneira geral, veja o Relatório de erros abaixo procurando pelo seu identificador de grupo. Em seguida, temos o Histórico de comentários com algumas anotações durante a revisão de código, testes manuais, etc.

2 Relatório de erros

2.1 Falhas de segmentação e Liberação dupla (*double free*)

As seguintes entradas (veja coluna `Test`) causam falha de segmentação (Res = 139) ou liberação dupla de memória (Res = 134) no programa. Para estes testes, sugiro que o grupo compile os fontes com -g além do -c para em seguida executar o valgrind com `--leak-check=full` conforme anteriormente explicado. Isso permitirá ao grupo identificar a origem do problema, resolvendo-o.

Grupo	Test	Res	Code	TEST.WORKED
GrupoN	kjl00	139	10	FALSE
GrupoN	kjl07	139	10	FALSE
GrupoN	kjl09	139	10	FALSE
GrupoN	kjl10	139	10	FALSE
GrupoN	kjl11	139	10	FALSE
GrupoN	kjl12	139	10	FALSE
GrupoN	kjl13	139	10	FALSE
GrupoN	kjl14	139	10	FALSE
GrupoN	kjl15	139	10	FALSE
GrupoN	kjl20	139	10	FALSE
GrupoN	kjl21	139	10	FALSE
GrupoN	kjl23	139	10	FALSE
GrupoN	kjl29	139	0	FALSE
GrupoO	kjl06	134	0	FALSE
GrupoO	kjl26	134	0	FALSE
GrupoO	kjl29	134	0	FALSE

2.2 Erros não são reconhecidos pelo programa

Os erros semânticos não reconhecidos abaixo acontecem quando o código do programa retorna zero, quando na realidade se esperava um erro semântico, com o código que aparece na tabela. Para facilitar a compreensão de onde o equívoco acontece, lista-se na coluna `Test` a entrada de teste utilizada para identificar o problema.

Grupo	Test	Res	Code	Erro

Continued on next page

Continued from previous page

Grupo	Test	Res	Code	Erro
GrupoC	kjl28	0	11	ERR_DECLARED
GrupoI	kjl08	0	11	ERR_DECLARED
GrupoK	kjl04	0	11	ERR_DECLARED
GrupoK	kjl27	0	11	ERR_DECLARED
GrupoK	kjl28	0	11	ERR_DECLARED
GrupoK	kjl30	0	11	ERR_DECLARED
GrupoN	kjl01	0	11	ERR_DECLARED
GrupoN	kjl02	0	11	ERR_DECLARED
GrupoN	kjl04	0	11	ERR_DECLARED
GrupoN	kjl05	0	11	ERR_DECLARED
GrupoN	kjl08	0	11	ERR_DECLARED
GrupoN	kjl16	0	10	ERR_UNDECLARED
GrupoN	kjl17	0	10	ERR_UNDECLARED
GrupoN	kjl18	0	11	ERR_DECLARED
GrupoN	kjl22	0	10	ERR_UNDECLARED
GrupoN	kjl24	0	11	ERR_DECLARED
GrupoN	kjl25	0	10	ERR_UNDECLARED
GrupoN	kjl27	0	11	ERR_DECLARED
GrupoN	kjl28	0	11	ERR_DECLARED
GrupoN	kjl30	0	11	ERR_DECLARED

2.3 Detecta erro sintático para entradas sintaticamente válidas

Para as entradas listadas nesta tabela, o programa detecta um erro sintático (código de retorno 1 na coluna `Res`) para entradas que são consideradas sintaticamente válidas de acordo com o definido nas etapas anteriores. Para fins de indicação, lista-se na coluna `Code` o erro *semântico* que deveria ter sido lançado para o teste.

Grupo	Test	Res	Code	TEST.WORKED

2.4 Detecta erro semântico em situações sem erro semântico esperado

O programa detecta erro semântico em testes onde nenhum erro semântico é esperado. Somente execuções que vão até o final sem falha de segmentação ou semelhantes são listadas.

Grupo	Test	Res	Code	TEST.WORKED
GrupoO	kjl19	11	0	FALSE

2.5 Detecta o erro semântico errado

O programa detecta um erro semântico diferente daquele esperado. Na tabela apresenta-se na coluna `Test` o identificador do teste onde o programa do grupo gera um código de erro (coluna `Res`) quando o esperado é o especificado na coluna `Code`. Somente execuções que vão até o final sem falha de segmentação ou semelhantes são listadas.

Grupo	Test	Res	Code	TEST.WORKED

3 Histórico de comentários

3.1 DONE GrupoA

3.1.1 E4

- Evitar de quebrar a linha na mensagem de erro.

☐ Na regra `atribuicao`, faltou definir a inferência para o nó da AST correspondente (não é necessário invocar `tipagemDado`, apenas impor o tipo de quem recebe o valor para o nó da AST =).

☐ Melhorar a indentação do arquivo `parser.y`. Por exemplo, colocar comandos de C em sequência na linha, como é feito nas regras gramaticais de expressões, pode tornar difícil a leitura do código.

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

3.1.2 E3

- Nenhum comentário.

3.1.3 E2

☒ Arquivo parser.y submetido na versão definitiva

☒ Poderiam documentar o comando `%locations` e o que ele faz

3.1.4 E1

☐ Para os tokens especiais, pode-se colocá-los todos na mesma regra

- E usar yytext[0] ao invés de explicitamente usar o literal

☐ Quebra de linha é considerado espaço em branco (pode-se colocar na mesma regra)

3.2 DONE GrupoB

3.2.1 E4

1. Normal
   - Agora compila.
   - ☐ Na hora de montar o pacote tgz, por favor, remover todos os arquivos "ocultos" que começam por ".". Eles não aparecem na saída do comando `ls`, mas o `tar` os vê e os inclui. Informar ao `tar` para não inclui-los.
   - ☐ Melhorar a indentação do arquivo `parser.y`. Por exemplo, colocar comandos de C em sequência na linha, como é feito em várias regras gramaticais, pode tornar difícil a leitura do código.
2. Preliminar
   - ☒ Não compila pois falta `SIZE_MAX`

3.2.2 E3

☐ Veja os pontos não marcados como feitos abaixo, na E1; entendo que eles ainda estão em aberto.

3.2.3 E2

☒ É necessário atualizar o arquivo `README.md` na medida que o projeto avança

- Por exemplo, a listagem da estrutura do projeto

3.2.4 E1

- Usando Makefile como um "wrapper" do CMake (só para deixar anotado)
- Boa documentação do projeto com o arquivo `README.md`

☒ Espaços ignorados podem ser aglutinados em uma única regra

- Quebras de linha são considerados espaços

☐ Na hora de montar o pacote tgz, por favor, remover todos os arquivos "ocultos" que começam por ".". Eles não aparecem na saída do comando `ls`, mas o `tar` os vê e os inclui. Informar ao `tar` para não inclui-los.

3.3 DONE GrupoC

3.3.1 E4

☐ Na regra de `atribuicao`, porque o grupo verifica se existem tipos incompatíveis se na realidade não existem compatíveis na especificação?

☒ Na regra `chamada_funcao`, a E4 explicita que nada deve ser feito para a chamada de função no que diz respeito aos tipos de dados. Por que o grupo então fez a definição do tipo do nó da chamada da função?

- O grupo conversou com o professor e explicou que fizeram isso como um bonus, um "a mais" na implementação pois sentiram falta da existência do tipo para a chamada de função pois esta faz parte de expressões.

☐ A implementação da "inferência" de tipos foi por copy-paste. Usar o conhecimento de Eng. Soft., encapsulando esse código em uma chamada de função, por exemplo.

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

3.3.2 E3

☐ A estrutura do valor léxico não é um ponteiro no `%union`. OK, mas atenção às próximas etapas pois o valor léxico deverá ser "copiado" (ao invés de ter seu ponteiro gerenciado).

- Grupo decidiu na E4 manter como não ponteiro

3.3.3 E2

☒ Funções "estranhas" (não usadas) no final do parser.y

- Exemplo, `parse_string`. Imagino que sejam para um uso alternativo de funcionamento do parser, visto que envolvem uma chamada à `yyparse`.

☒ No arquivo main.c, o include do `parser.tab.h` possui um caminho relativo. Em geral isso não é uma boa prática, sendo preferível informar ao compilador onde procurar arquivos de cabeçalhos via o parâmetro `-I` (neste caso, poderia ser algo como `-I./obj/` em algum lugar do `Makefile`).

3.3.4 E1

☒ O arquivo scanner.l, contendo código, normalmente fica em diretórios `src`

☒ O arquivo Makefile não segue a ideia de compilação separada por arquivo, algo em geral benéfico até para projetos pequenos.

- O alvo `test` não funciona, provavelmente porque o conteúdo de `src/testing` foi removido do pacote
- Pode-se criar uma arquivo `Makefile.alt` somente para testes
  - Removê-lo do tgz na hora de submeter

3.4 DONE GrupoD

3.4.1 E4

☐ Melhorar a indentação do arquivo `parser.y`. Por exemplo, colocar comandos de C em sequência na linha, como é feito em várias regras gramaticais, pode tornar difícil a leitura do código.

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

☐

3.4.2 E3

1. Recuperação

   ☐ A estrutura do valor léxico não é um ponteiro no `%union`. OK, mas atenção às próximas etapas pois o valor léxico deverá ser "copiado" (ao invés de ter seu ponteiro gerenciado).
   - Grupo decidiu por manter assim na E4.

   ☐ Veja os pontos não marcados como feitos abaixo, na E1; entendo que eles ainda estão em aberto.
   - Grupo decidiu por manter sem modificar na E4

2. Normal

   ☐ Praticamente todos os testes falham.
   - Ao invés de gerar uma única árvore, várias subárvores aparecem na saída

   ☐ A estrutura do valor léxico não é um ponteiro no `%union`. OK, mas atenção às próximas etapas pois o valor léxico deverá ser "copiado" (ao invés de ter seu ponteiro gerenciado).

   ☐ Veja os pontos não marcados como feitos abaixo, na E1; entendo que eles ainda estão em aberto.

3.4.3 E2

☒ A regra `expressao` não segue a especificação da E2 pois não implementa precedência de operadores (Sec 3.4 da E2). Para que a precedência possa ser implementada, precisas criar "níveis", por exemplo, no primeiro nível é o or, depois o and, e assim por diante. Seria algo assim:

```
expressao: expressao TK_OC_OR exp1 | exp1
exp1: exp1 TK_OC_AND exp2 | exp2
exp2: ...
```

Podemos trocar uma ideia caso ainda não tiverem entendido.

3.4.4 E1

☐ Para os tokens especiais, pode-se colocá-los todos na mesma regra

- E usar yytext[0] ao invés de explicitamente usar o literal
- Grupo decidiu por manter assim na E4

☐ Melhorar o makefile para que se possa usufruir de um sistema de compilação que permita compilação parcial dos vários fontes do projeto.

- Grupo decidiu por manter assim na E4

3.5 DONE GrupoE

3.5.1 E4

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

3.5.2 E3

☒ A estrutura do valor léxico não é um ponteiro no `%union`. OK, mas atenção às próximas etapas pois o valor léxico deverá ser "copiado" (ao invés de ter seu ponteiro gerenciado).

3.5.3 E2

☒ O que é a "metodologia de tdd"? (comentário no `scanner.l`)

- Não respondido.

☒ A regra `expressao` não segue a especificação da E2 pois não implementa precedência de operadores (Sec 3.4 da E2). Para que a precedência possa ser implementada, precisas criar "níveis", por exemplo, no primeiro nível é o or, depois o and, e assim por diante. Seria algo assim:

```
expressao: expressao TK_OC_OR exp1 | exp1
exp1: exp1 TK_OC_AND exp2 | exp2
exp2: ...
```

Podemos trocar uma ideia caso ainda não tiverem entendido.

3.5.4 E1

☒ Para os tokens especiais, pode-se colocá-los todos na mesma regra

- E usar yytext[0] ao invés de explicitamente usar o literal

3.6 DONE GrupoF

3.6.1 E4

☐ As mensagens de erro poderiam fazer referência aos lexemas dos símbolos envolvidos no erro, facilitando a identificação. Por exemplo, em kjl00, qual é o símbolo que não foi declarado?

☐ Na regra `atribuicao`, faltou definir a inferência para o nó da AST correspondente.

☐ Melhorar a indentação do arquivo `parser.y`. Por exemplo, colocar comandos de C em sequência na linha, como é feito nas regras gramaticais de expressões, pode tornar difícil a leitura do código.

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

3.6.2 E3

1. Recuperação
   - Nenhum comentário.
2. Normal
   - Muitos testes falham.
     - Labels errados (problemas de copy/paste?)
       * Por exemplo no igual-igual
     - Mas também temos erros mais graves de uso da `asd`.

3.6.3 E2

- Vários testes falham. Olhando rapidamente, parece que o problema tem a ver com a `expressao` que deve virar um `literal` ou apenas com `literal`.

3.6.4 E1

☐ O Makefile não segue a filosofia geral para construção de projetos, pois possui listagens de código nas dependências e não possui uma regra de compilação intermediária de código objeto que permite a compilação parcial do projeto.

☒ Os espaços podem ser aglutinados em uma única regra

☒ Para os tokens especiais, pode-se colocá-los todos na mesma regra

  – E usar yytext[0] ao invés de explicitamente usar o literal

3.7 DONE GrupoG

3.7.1 E4

☐ Na regra `chamada_funcao`, a E4 explicita que nada deve ser feito para a chamada de função no que diz respeito aos tipos de dados. Por que o grupo então fez a definição do tipo do nó da chamada da função?

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

☐ Evitar uso de variáveis globais (tais como a `desired_kind`), ainda que funcione. Sempre há uma forma melhor.

3.7.2 E3

☒ A estrutura do valor léxico não é um ponteiro no `%union`. OK, mas atenção às próximas etapas pois o valor léxico deverá ser "copiado" (ao invés de ter seu ponteiro gerenciado).

3.7.3 E2

- Nenhum comentário.

3.7.4 E1

☒ Arquivos devem estar na raiz

☒ Normalmente evita-se de #include com um caminho relativo (ou absoluto)

  – No caso farias somente #include "tokens.h", instruindo o compilador (com -I) a procurar os cabeçalhos em determinado lugar

☒ Para os tokens especiais, pode-se colocá-los todos na mesma regra

  – E usar yytext[0] ao invés de explicitamente usar o literal

☒ A regra do barra-enne é redundante com a classe [:space:]

☒ Boa organização em subdiretórios, mas o Makefile precisa melhorar visto que não é uma boa prática em receitas misturar entradas .c e .o como no alvo $(ETAPA). Além disso, a compilação de .o pode ser via regra única por intermédio de wildcards, conforme visto no tutorial indicado.

☒ Arquivos `deliver.sh`, `tester.py`, `tests` podem ser omitidos do tgz

3.8 DONE GrupoH

3.8.1 E4

☐ Na regra `atribuicao`, faltou definir a inferência para o nó da AST correspondente.

☐ Na regra `chamadaFuncao`, a E4 explicita que nada deve ser feito para a chamada de função do que diz respeito aos tipos de dados. Por que o grupo então fez a definição do tipo do nó da chamada da função?

3.8.2 E3

☐ typedef é seu amigo para o tipo da árvore

  – Grupo decidiu na E4 ainda não usar typedef

3.8.3 E2

- Nenhum comentário.

3.8.4 E1

☐ O caractere barra-ene está incluso na classe `[:blank:]`, portanto temos uma regra redundante

☒ O Makefile não segue a filosofia geral para construção de projetos, pois possui listagens de código nas dependências e não possui uma regra de compilação intermediária de código objeto que permite a compilação parcial do projeto.

3.9 DONE GrupoI

3.9.1 E4

1. Em atraso
   - Submetido
   - Comentários feitos para a E3 não executados
2. No prazo
   - Não submetido.

3.9.2 E3

1. Em atraso
   ☐ O valor léxico (`yylval.valor_lexico`), conforme E3, deve ser especificado apenas para literais e identificadores. Na solução submetido, temos valor léxico para vários outros elementos desnecessários (palavras-reservadas, operadores simples e compostos, etc).
   ☐ Na hora de montar o pacote tgz, por favor, remover todos os arquivos "ocultos" que começam por ".". Eles não aparecem na saída do comando `ls`, mas o `tar` os vê e os inclui. Informar ao `tar` para não inclui-los.
   ☐ A estrutura do valor léxico não é um ponteiro no `%union`. OK, mas atenção às próximas etapas pois o valor léxico deverá ser "copiado" (ao invés de ter seu ponteiro gerenciado).
2. Normal
   - Não submetido no prazo

3.9.3 E2

- Arquivos devem estar na raiz
  - O peso deste comentário aumentou

3.9.4 E1

☐ Arquivos devem estar na raiz

☒ Deve-se evitar colocar a implementação de funções no cabeçalho do scanner, priorizando a última seção do arquivo ou em arquivos suplementos.

☒ Ao invés de implementar yywrap, pode-se usar a opção para desabilitar essa funcionalidade.

- [/] Alvos e receitas do makefile são majoritariamente manuais, sem wildcards. O makefile pode ficar bem mais sucinto se empregar os conhecimentos do tutorial indicado.
  - Além disto, possui regras específicas da etapa1, mesmo sabendo que temos outras etapas por vir.

3.10 DONE GrupoJ

3.10.1 E4

☐ Na regra `atribuicao_variavel`, faltou definir o tipo de dado para o nó da AST correspondente. Além disso, não é necessário validar pois aqui todos os tipos são compatíveis entre si, e, ainda mais, o tipo de dado da atribuição é imposta pelo tipo do identificador.

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

3.10.2 E3

☐ Na hora de montar o pacote tgz, por favor, remover todos os arquivos "ocultos" que começam por ".". Eles não aparecem na saída do comando `ls`, mas o `tar` os vê e os inclui. Informar ao `tar` para não inclui-los.

☒ Veja os pontos não marcados como feitos abaixo, tanto na E2 quanto na E1; entendo que eles ainda estão em aberto.

3.10.3 E2

- Preliminar: calculou a coluna do erro sintático!

☒ No arquivo main.c, o include do `parser.tab.h` possui um caminho relativo. Em geral isso não é uma boa prática, sendo preferível informar ao compilador onde procurar arquivos de cabeçalhos via o parâmetro `-I` (neste caso, poderia ser algo como `-I./obj/` em algum lugar do `Makefile`).

☒ O arquivo `Makefile` evolui bastante, mas de uma maneira geral ele ficou muito complexo (veja o tamanho). Poderia ficar bem mais simples, sobretudo pela característica ainda pequena de nosso projeto.

☒ Veja os pontos não marcados como feitos abaixo; entendo que eles ainda estão em aberto.

3.10.4 E1

☒ As ações associadas às regras estão com identação diferente, algumas alinhadas outras não, no arquivo scanner.l

☒ O makefile parece ter código boilerplate que não se aplica neste projeto, pois ele vê se existe um subdiretório src, adaptando-se em função. Se o grupo não tem a intenção de organizar em subdiretórios, sugiro simplificar o Makefile.

☒ Não se usa a filosofia de compilação parcial dos arquivos (-c), ou seja, sempre se compila tudo novamente.

☒ A variável tar file pode ser definida a partir do nome de binary.

3.11 DONE GrupoK

3.11.1 E4

☐ Melhorar a indentação do arquivo `parser.y`. Por exemplo, colocar comandos de C em sequência na linha, como é feito em todas as regras gramaticais de expressões, pode tornar difícil a leitura do código.

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

☐ Declarar as funções no escopo global, confirmar que elas estão sendo corretamente verificadas (kjl04).

3.11.2 E3

☐ A estrutura do valor léxico não é um ponteiro no `%union`. OK, mas atenção às próximas etapas pois o valor léxico deverá ser "copiado" (ao invés de ter seu ponteiro gerenciado).

3.11.3 E2

☒ Preliminar: arquivos devem estar na raiz

3.11.4 E1

☐ Não há necessidade de stdio.h no arquivo scanner.l

☒ Melhorar o makefile para que se possa usufruir de um sistema de compilação que permita compilação parcial dos vários fontes do projeto.

3.12 DONE GrupoL

3.12.1 E4

- Entregue com 2hs de atraso. Entregar no prazo.

☐ Na regra `comando_atribuicao`, faltou definir a inferência de tipo para o nó da AST correspondente.

☐ Melhorar a indentação do arquivo `parser.y`. Por exemplo, colocar comandos de C em sequência na linha, como é feito nas regras gramaticais de expressões, pode tornar difícil a leitura do código.

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

3.12.2 E3

☒ Veja os pontos não marcados como feitos abaixo, na E1; entendo que eles ainda estão em aberto.

3.12.3 E2

- Nenhum comentário.

3.12.4 E1

☒ Normalmente o ; nos comandos em C ficam imediatamente após a último token do comando, sem espaços.

☒ Não há necessidade de incluir stdio.h. Além disso, cabeçalhos de bibliotecas de sistema são incluídas com <stdio.h> ao invés de "stdio.h".

☒ Melhorar o makefile para que se possa usufruir de um sistema de compilação que permita compilação parcial dos vários fontes do projeto.

☒ Evitar de empacotar o diretório `testes` (e os arquivos `quero.*\.sh`).

3.13 DONE GrupoM

3.13.1 E4

☐ Na regra `function_call`, a E4 explicita que nada deve ser feito para a chamada de função no que diz respeito aos tipos de dados. Por que o grupo então fez a definição do tipo do nó da chamada da função?

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

3.13.2 E3

☐ A estrutura do valor léxico não é um ponteiro no `%union`. OK, mas atenção às próximas etapas pois o valor léxico deverá ser "copiado" (ao invés de ter seu ponteiro gerenciado).

- Grupo decidiu na E4 manter tal qual.

☒ Nos comentários que categorizam as regras gramaticais, usar texto não em uppercase total.

☒ Veja os pontos não marcados como feitos abaixo, na E2; entendo que eles ainda estão em aberto.

3.13.3 E2

☒ Preliminar: usou o comando %left, em desacordo com a especificação E2

☒ Usar wildcards do makefile para simplificar as regras

3.13.4 E1

☒ Melhorar o makefile para que se possa usufruir de um sistema de compilação que permita compilação parcial dos vários fontes do projeto.

☒ Não há necessidade de incluir stdio.h.

☒ Os espaços ignorados podem ser aglutinados em uma única regra

3.14 DONE GrupoN

3.14.1 E4

1. Recuperação
   - Não submetido
2. Normal
   - ☐ Falha de segmentação em vários testes
   - ☐ Nenhum erro semântico detectado
   - ☐ Na hora de montar o pacote tgz, por favor, remover todos os arquivos "ocultos" que começam por ".". Eles não aparecem na saída do comando `ls`, mas o `tar` os vê e os inclui. Informar ao `tar` para não inclui-los. (segundo aviso)
   - ☐ O professor identificou que muito pouco código é referente à E4, quando comparando, por exemplo, contra a E3 do grupo.
   - ☐ Foi criado um módulo chamado `comp_utils` que agora contém código da parte da AST (E3). A sugestão é manter um módulo bem caracterizado pois algo "utils" é genérico sem funcionalidade específica. Essas mudanças dificultam também a avaliação pelo professor. :-(
   - ☐ Na regra `func`, atenção porque o identificador da função está sendo incluído na tabela local e não na global, onde deveria ser. Antes de `pushLocalTable`, deve haver inserção do `TK_IDENTIFICADOR` na tabela global. Sugiro reverificar os pontos de empilhamento e desempilhamento.
   - ☐ Não foi encontrada instrumentação da AST para inferência de tipos (o nó da AST não possui campo de tipo de dado).
   - ☐ A estrutura `ast_token` também tem campos não solicitados pela especificação. Talvez pudessem fazer uma limpeza de coisas desnecessárias.

3.14.2 E3

1. Recuperação

   ☐ O valor léxico (`yylval.valor_lexico`), conforme E3, deve ser especificado apenas para literais e identificadores. Na solução submetido, temos valor léxico para vários outros elementos desnecessários, como para operadores simples e compostos.

   ☐ Veja os pontos não marcados como feitos abaixo, na E2; entendo que eles ainda estão em aberto.

2. Normal

   ☐ Falha de segmentação em praticamente todos os testes.

   ☐ O valor léxico (`yylval.valor_lexico`), conforme E3, deve ser especificado apenas para literais e identificadores. Na solução submetido, temos valor léxico para vários outros elementos desnecessários (palavras-reservadas, operadores simples e compostos, etc).

   ☐ Obtive um "warning: type clash on default action" na linha 132 do parser.y. Isso ocorre porque `cmdblck` tem tipo, e nenhuma ação foi incluída ali para definir o valor desse NT. A ação poderia ser algo como $$ = $2.

   ☐ Veja os pontos não marcados como feitos abaixo, na E2; entendo que eles ainda estão em aberto.

3.14.3 E2

☐ Melhorar o makefile para que se possa usufruir de um sistema de compilação que permita compilação parcial dos vários fontes do projeto.

3.14.4 E1

☒ Na hora de montar o pacote tgz, por favor, remover todos os arquivos "ocultos" que começam por ".". Eles não aparecem na saída do comando `ls`, mas o `tar` os vê e os inclui. Informar ao `tar` para não inclui-los.

☒ Não temos aspas simples

☒ Não temos comentários multilinha portanto não há necessidade de %x

3.15 DONE GrupoO

3.15.1 E4

☐ Na inferência de tipos das expressões, emprega-se a `typeInfer`, que recebe apenas um nó da AST e faz a inferência baseado nos filhos. O problema é que naquelas regras, essa função é chamada apenas com o terceiro filho como parâmetro. Ou seja, a inferência é feita somente naquele filho, e não no nó que está sendo criado. A inferência do nó que está sendo criada é portanto postergada. Qual a razão de se implementar assim?

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

3.15.2 E3

☒ Veja os pontos não marcados como feitos abaixo, na E1; entendo que eles ainda estão em aberto.

3.15.3 E2

☒ A regra `expressao` não segue a especificação da E2 pois não implementa precedência de operadores (Sec 3.4 da E2). Para que a precedência possa ser implementada, precisas criar "níveis", por exemplo, no primeiro nível é o or, depois o and, e assim por diante. Seria algo assim:

```
expressao: expressao TK_OC_OR exp1 | exp1
exp1: exp1 TK_OC_AND exp2 | exp2
exp2: ...
```

Podemos trocar uma ideia caso ainda não tiverem entendido.

☒ Melhorar a identação das receitas do makefile, um tab é o suficiente.

☒ Melhorar o makefile para que se possa usufruir de um sistema de compilação que permita compilação parcial dos vários fontes do projeto.

3.15.4 E1

☒ Os espaços ignorados podem ser aglutinados em uma única regra

- Inclusive o barra-enne poderia ser aglutinado visto que trata-se de um espaço também

☒ Comentários são legais, mas melhor se estiverem não todos em maiúscula

- Para não confundir com constantes do código

3.16 DONE GrupoP

3.16.1 E4

☐ Poderia implementar uma função para a inferência, evitando cópia de código.

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

3.16.2 E3

☒ Temos três campos em `%union`, sendo um `asd_tree_p`. Para que ele serve? De acordo com a especificação E3, seriam somente necessários os dois primeiros campos.

☒ A estrutura do valor léxico não é um ponteiro no `%union`. OK, mas atenção às próximas etapas pois o valor léxico deverá ser "copiado" (ao invés de ter seu ponteiro gerenciado).

☐ Veja os pontos não marcados como feitos abaixo, na E1; entendo que eles ainda estão em aberto.

3.16.3 E2

☒ Melhor identificar que o número que aparece no relatório de erro é um número de linha

☒ Documentar melhor as regras da gramáticas, as agrupando e colocando espaços entre categorias de regras, de maneira a facilitar a leitura da gramática.

3.16.4 E1

☐ Melhorar o makefile para que se possa usufruir de um sistema de compilação que permita compilação parcial dos vários fontes do projeto.

- Nada novo aqui na E4

3.17 GrupoQ

3.17.1 E4

- Não submetido

3.17.2 E3

- Não submetido

3.17.3 E2

- Não submetido

3.17.4 E1

- Não submetido

3.18 DONE GrupoR

3.18.1 E4

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

3.18.2 E3

☒ A estrutura do valor léxico não é um ponteiro no `%union`. OK, mas atenção às próximas etapas pois o valor léxico deverá ser "copiado" (ao invés de ter seu ponteiro gerenciado).

3.18.3 E2

☒ Preliminar: melhor identificar que o número que aparece no relatório de erro é um número de linha

3.18.4 E1

☒ Na função `get_line_number`, corrigir a identação.

☒ Melhorar o makefile para que se possa usufruir de um sistema de compilação que permita compilação parcial dos vários fontes do projeto.

- Ter um alvo que empregue o parâmetro `-c`

3.19 DONE GrupoS

3.19.1 E4

- Boa organização do código em módulos

☐ Minha sugestão seria para corrigir a indentação, padronizando-a, sobretudo no arquivo parser.y. Por exemplo, na regra `atribuicao`, temos um problema de indentação (em outras regras com comandos if com else if, e if com else também).

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

3.19.2 E3

☒ A estrutura do valor léxico não é um ponteiro no `%union`. OK, mas atenção às próximas etapas pois o valor léxico deverá ser "copiado" (ao invés de ter seu ponteiro gerenciado).

3.19.3 E2

☒ Nos comentários que categorizam as regras gramaticais, usar texto não em uppercase total.

3.19.4 E1

☒ Ignorar também o caractere tab com barra-t

☒ Aglutinar caracteres ignorados (espaços) em uma única regra

☒ Melhorar o makefile para que se possa usufruir de um sistema de compilação que permita compilação parcial dos vários fontes do projeto.

3.20 DONE GrupoT

3.20.1 E4

☐ Usar a flag -I do compilador ao invés de incluir com caminhos relativos tal como "`../include/data_structures.h`".

☐ Melhorar a indentação do arquivo `parser.y`. Por exemplo, colocar comandos de C em sequência na linha, como é feito nas regras gramaticais de expressões, pode tornar difícil a leitura do código.

☐ As ações de várias regras possuem código repetido, isso poderia ser organizado em funções (Eng. Soft.) de maneira a tornar o programa mais legível.

3.20.2 E3

☒ Veja os pontos não marcados como feitos abaixo, na E2; entendo que eles ainda estão em aberto.

3.20.3 E2

☒ Remover o arquivo tokens.h conforme recomendado na especificação E2

☒ No makefile, a compilação de .o pode ser via regra única por intermédio de wildcards, conforme visto no tutorial indicado.

3.20.4 E1

☒ O arquivo scanner.l, contendo código, normalmente fica em diretórios `src`

☒ O arquivo Makefile não segue a ideia de compilação separada por arquivo, algo em geral benéfico até para projetos pequenos.

☒ Para os tokens especiais, pode-se colocá-los todos na mesma regra

- E usar yytext[0] ao invés de explicitamente usar o literal

3.21 GrupoZ

3.21.1 E4

- Não submetido.

3.21.2 E3

1. Submissão em atraso

   ☐ A estrutura do valor léxico não é um ponteiro no `%union`. OK, mas atenção às próximas etapas pois o valor léxico deverá ser "copiado" (ao invés de ter seu ponteiro gerenciado).

3.21.3 E2

☒ Retomo o comentário deixado pelo grupo "TODO: Check for precedencia"

- Realmente ficou faltando.

3.21.4 E1

☒ Melhorar o makefile para que se possa usufruir de um sistema de compilação que permita compilação parcial dos vários fontes do projeto.

4 Estatísticas

4.1 Maior quantidade de erros esperados

O código zero na coluna “Erro” indica situação onde não há erro.

Erro	n	Porcento
ERR-UNDECLARED	16	51.61
ERR-DECLARED	10	32.26
0	5	16.13

4.2 Quais testes foram mais errados pelos grupos?

Test	Quantos.Grupos.Erraram	Erro.Esperado
kjl28	3	ERR-DECLARED

5 Pesos

Grupo	E4.P
GrupoA	1
GrupoB	1
GrupoC	1
GrupoD	1
GrupoE	1
GrupoF	1
GrupoG	1
GrupoH	1
GrupoI	0.8
GrupoJ	1
GrupoK	1
GrupoL	1
GrupoM	1
GrupoN	1
GrupoO	1
GrupoP	1
GrupoR	1
GrupoS	1
GrupoT	1

6 Final

Grupo	Etapa	E4.O	E4.S	E4.P
GrupoA	E4	10	8.97	1
GrupoB	E4	10	9.45	1
GrupoC	E4	9.68	8.97	1
GrupoD	E4	10	9.45	1
GrupoE	E4	10	9.83	1
GrupoF	E4	10	8.22	1
GrupoG	E4	10	9.18	1
GrupoH	E4	10	8.43	1
GrupoI	E4	9.68	9.45	0.8
GrupoJ	E4	10	8.97	1
GrupoK	E4	8.71	8.35	1
GrupoL	E4	10	9.45	1
GrupoM	E4	10	9.18	1
GrupoN	E4	1.29	1.85	1
GrupoO	E4	8.71	7.3	1
GrupoP	E4	10	9.72	1
GrupoR	E4	10	9.78	1
GrupoS	E4	10	9.83	1
GrupoT	E4	10	9.72	1

7 Recuperação

Grupos em recuperação da E4:

- **Apenas grupos que entregaram no prazo** conforme regramentos
- Em Moodle já configurado para tal, use o link "Recuperação E4"
- Política de recuperação ativada (vejam regras gerais)

$$\frac{x}{\text{GrupoN}}$$