COMP LabBook 2024 2 - E5

Lucas M. Schnorr

December 19, 2024

Contents

1 Introdução

1.1 Metodologia de avaliação automática

O teste automático da E5 faz uso do simulador ILOC. Funciona assim:

1.1.1 Parte 1 (Execução)

1. São gerados posições aleatórias tanto para o início da pilha quanto para o início do segmento de dados. Assim:

   ```
   STACK=$(($RANDOM + 100000))
   DATA=$(($STACK / 2))
   ```

   Esses valores são registrados no log nas linhas que tem "stack" e "data". Dessa forma, ainda que os testes para todos os grupos sejam os mesmos, os valores computadores pelos códigos ILOC gerados deverão ficar em posições diferentes da "memória".

2. Em seguida, se lança a geração do código ILOC assim:

   ```
   timeout 3 ./etapa5 < $test > /tmp/saida.iloc
   ```

   O conteúdo de saida.iloc é registrado no log nas linhas que tem "input".

3. Em seguida, se lança o simulador para simular o código ILOC:

   ```
   timeout 3 ${ILOCSIM} \
                          --stack $STACK \
                          --data $DATA \
                          -s \
                          -x < /tmp/saida.iloc > /tmp/saida.sim
   ```

   Vejam que o modo strict é usado, assim como se força a localização da pilha e do segmento de dados. O conteúdo de saida.sim é salvo no logo através das linhas "output".

4. Para facilitar a análise dos resultados, é feito um parse do saida.sim, assim:

   ```
   cat /tmp/saida.sim | \
        awk '/    memory    value/,0' | \
                       grep -v memory | \
                       awk '{print $1 "|" $2}'
   ```

   O resultado mostra efetivamente o estado da memória após a simulação do código ILOC gerado pelo compilador. É sobre esse estado que é feito o cálculo da nota objetiva.

1.1.2 Parte 2 (Definição da resposta correta)

Como as posições de pilha/dados de cada teste são geradas aleatoriamente, nós precisamos definir a resposta correta em função do valor de início da pilha/dados de cada teste. Isso é feito através de um pós-processamento do arquivo de log e do conteúdo da tabela `e5_correct.csv`. Após esse processamento, temos a tabela `e5_detalhes.csv` que mostra, para cada teste, em qual posição de memória deve ficar o valor informado. Para o teste funcionar, precisamos verificar se o valor especificado na coluna **Value** encontra-se na posição de memória esperada comparando **Address.cor** (correto) com **Address.mem** (observado, gerado pelo grupo).

1.1.3 Parte 3 (Decisão da resposta correta)

Para o teste ser considerado como correto, existem duas condições: (1) o valor a ser observado no endereço de memória deve ser aquele esperado; (2) a distância entre o endereço correto e o endereço de memória deve ser menor ou igual à 512 (para permitir uma certa flexibilidade na geração de código). O resultado dessa decisão, para cada teste, é registrado na coluna **Correto** na tabela `e5_detalhes_resultado.csv`. A coluna **Correto** deve estar com o valor **TRUE**. Quando **FALSE** quer dizer que as condições acima falharam. As condições para falha podem ser por que o programa deu segfault, ou o código ILOC não foi gerado, ou foi gerado mas o simulador não reconheceu, enfim, qualquer outro cenário que não permitiu chegar até a possibilidade da verificação acima.

1.2 Resultados corretos

Espera-se que, para cada teste explicitado na coluna **Test**, o programa ILOC gerado pelo compilador coloque o valor especificado na coluna **Value** no local especificado na coluna **Type**. Os dois locais conhecidos são a pilha (Stack) e o segmento de dados (Data).

Test	Value	Type
abc00	456	Stack
abc00	98	Stack
abc01	-3	Stack
abc01	432	Stack
abc02	123	Stack
abc02	357	Stack
abc03	1	Stack
abc03	393	Stack
abc04	5	Stack
abc04	615	Stack
abc06	0	Stack
abc06	1	Stack
abc06	2	Stack
abc06	3	Stack
abc06	290	Stack
abc07	0	Stack
abc07	1	Stack
abc07	2	Stack
abc07	3	Stack
abc07	322	Stack
ijk00	923	Stack
ijk00	1379	Stack
ijk00	456	Stack
ijk01	923	Stack
ijk01	467	Stack
ijk01	456	Stack
ijk02	20	Stack
ijk02	30	Stack
ijk02	600	Stack
ijk03	12	Stack
ijk03	3	Stack
ijk03	4	Stack
ijk04	30	Stack
ijk04	46	Stack
ijk04	76	Stack
ijk05	-2	Stack
ijk05	-18	Stack
ijk05	16	Stack
ijk06	25	Stack
ijk06	5	Stack
ijk06	12	Stack
ijk07	45	Stack
ijk07	135	Stack

Continued on next page

Continued from previous page

Test	Value	Type
ijk07	6075	Stack
ijk08	55	Stack
ijk08	15	Stack
ijk08	3	Stack
ijk09	46	Stack
ijk09	15	Stack
ijk09	3	Stack
ijk10	300	Stack
ijk10	400	Stack
ijk10	2	Stack
ijk11	800	Stack
ijk11	400	Stack
ijk11	2	Stack
ijk12	20	Stack
ijk13	2	Stack
ijk14	393	Stack
ijk14	394	Stack
ijk15	-393	Stack
ijk15	393	Stack
ijk16	0	Stack
ijk16	1	Stack
ijk17	109	Stack
ijk17	0	Stack
ijk17	1	Stack
ijk18	109	Stack
ijk18	1	Stack
ijk18	2	Stack
ijk19	109	Stack
ijk19	1	Stack
ijk19	4	Stack
ijk20	109	Stack
ijk20	1	Stack
ijk20	5	Stack
ijk21	109	Stack
ijk21	108	Stack
ijk21	0	Stack
ijk21	5	Stack
ijk22	109	Stack
ijk22	108	Stack
ijk22	1	Stack
ijk22	6	Stack
ijk23	109	Stack
ijk23	0	Stack
ijk23	5	Stack
ijk24	1	Stack
ijk24	6	Stack

Esta é a quantidade de testes:

```
89
```

2 Relatório de erros gerais

2.1 Timeouts da execução do código ILOC gerado

Uma das etapas da avaliação automática é a simulação do código ILOC gerado pelo compilador implementado pelo grupo, para uma data entrada. Na tabela abaixo estão listados casos onde o código ILOC gerado pelo grupo, para a entrada fornecida ao compilador especificada na coluna `Test`, não termina. Ou seja, o simulador ficaria para sempre executando o código ILOC sem terminar. A razão do "não terminar" em geral acontece por conta de algum laço infinito não intencional que acabou sendo gerado. Sugere-se a execução do simulador com a opção `-t` (de rastreamento das instruções executadas) para entender o que exatamente esta acontecendo. Todos os testes que dão timeout nessa fase de simulação são portanto considerado falhos por não permitir a realização da verificação automática da funcionalidade esperada.

Group	Test

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

A primeira etapa de avaliação automática é a execução do compilador do grupo com uma data entrada. Estão listados os casos de falha de segmentação quando as entradas identificadas pela coluna `Test` da tabela abaixo são fornecidas para o compilador do grupo. Como o compilador é então incapaz de gerar ILOC, estes testes são considerados falhos.

Group	Test
GrupoJ	abc00
GrupoJ	abc01
GrupoJ	abc02
GrupoJ	abc03
GrupoJ	abc04
GrupoJ	abc06
GrupoJ	abc07
GrupoJ	ijk14
GrupoJ	ijk16

2.3 Compilador não executa corretamenta (retorna *non-zero*)

Todas as entradas utilizadas nesta avaliação estão lexicalmente, sintaticamente e semanticamente válidas. Portanto, erros léxicos, sintáticos e semânticos não são esperados. O compilador do grupo detecta erros desses tipos quando as entradas especificadas na coluna `Test` são fornecidos em entrada. Estes são efeitos de não correções de situações muito provavelmente já identificadas em etapas anteriores, ou incluídas de maneira inadvertida durante a implementação desta etapa.

Group	Test

2.4 Simulador não reconhece as instruções ILOC geradas

Ao simular o código ILOC gerado pelo compilador do grupo, o `ilocsim` relata erro sintático nas instruções geradas. Na tabela abaixo estão listados tais casos para quando é gerado código ILOC para a entrada fornecida ao compilador identificada na coluna `Test`.

Group	Test	Content

2.5 O código ILOC gerado não faz o que o teste estimulava

Cada teste especificado na coluna `Test` tem um efeito, registrado nas variáveis locais ou globais. Este efeito pode ser oriundo de uma simples atribuição, de uma expressão aritmética, ou em função do controle de fluxo e chamadas de funções. Por exemplo, no teste `ijk01`, espera-se que o valor `1379`, resultado da soma, seja encontrado em um endereço na pilha. Na tabela abaixo, nós temos os testes que foram considerados errados porque o código ILOC gerado pelo compilador do grupo foi incapaz de gerar o valor especificado na coluna `Value` próximo ao endereço especificado em `Address.cor`, para o tipo de segmento (pilha ou dados) especificado na coluna `Type`. São listados apenas 20 linhas do arquivo `e5_detalhes_resultado.csv`; consulte tal arquivo para a totalidade deste relatório de erros, ou veja os comentários abaixo onde os erros são apresentados por grupo.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto
GrupoJ	abc06	3	Stack	119999			FALSE
GrupoJ	abc01	432	Stack	120407			FALSE
GrupoJ	abc03	1	Stack	109840			FALSE
GrupoJ	abc00	98	Stack	127087			FALSE
GrupoJ	abc00	456	Stack	127087			FALSE
GrupoJ	abc06	0	Stack	119999			FALSE
GrupoJ	ijk16	1	Stack	111081			FALSE
GrupoJ	ijk14	393	Stack	130070			FALSE
GrupoJ	abc02	123	Stack	131895			FALSE
GrupoJ	abc07	0	Stack	110696			FALSE
GrupoJ	abc04	615	Stack	114029			FALSE
GrupoJ	abc02	357	Stack	131895			FALSE
GrupoJ	abc07	2	Stack	110696			FALSE
GrupoJ	ijk14	394	Stack	130070			FALSE
GrupoJ	abc04	5	Stack	114029			FALSE
GrupoJ	abc06	290	Stack	119999			FALSE
GrupoJ	abc06	1	Stack	119999			FALSE
GrupoJ	abc07	1	Stack	110696			FALSE
GrupoJ	abc07	3	Stack	110696			FALSE
GrupoJ	abc03	393	Stack	109840			FALSE

3 Histórico de comentários

3.1 DONE GrupoA

3.1.1 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.1.2 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.

3.1.3 E3

- Nenhum comentário.

3.1.4 E2

☒ Arquivo parser.y submetido na versão definitiva

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

3.1.5 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 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.2.2 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`

3.2.3 E3

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

3.2.4 E2

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

  – Por exemplo, a listagem da estrutura do projeto

3.2.5 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 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.3.2 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.

3.3.3 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

3.3.4 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`).

3.3.5 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 E5

- Sem comentários

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.4.2 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.

3.4.3 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.

3.4.4 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.

3.4.5 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 E5

- Sem comentários

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.5.2 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.

3.5.3 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).

3.5.4 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.

3.5.5 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 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.6.2 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.

3.6.3 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`.

3.6.4 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`.

3.6.5 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 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.7.2 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.

3.7.3 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).

3.7.4 E2

- Nenhum comentário.

3.7.5 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 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.8.2 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?

3.8.3 E3

☐ typedef é seu amigo para o tipo da árvore

  – Grupo decidiu na E4 ainda não usar typedef

3.8.4 E2

- Nenhum comentário.

3.8.5 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 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.9.2 E4

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

3.9.3 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

3.9.4 E2

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

3.9.5 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.

Continued on next page

Continued from previous page

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.10 DONE GrupoJ

3.10.1 E5

☐ problemas nas expressões aritméticas, algumas instruções usam temporários que nunca foram definidos.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto
GrupoJ	abc00	98	Stack	127087			FALSE
GrupoJ	abc00	456	Stack	127087			FALSE
GrupoJ	abc01	-3	Stack	120407			FALSE
GrupoJ	abc01	432	Stack	120407			FALSE
GrupoJ	abc02	123	Stack	131895			FALSE
GrupoJ	abc02	357	Stack	131895			FALSE
GrupoJ	abc03	1	Stack	109840			FALSE
GrupoJ	abc03	393	Stack	109840			FALSE
GrupoJ	abc04	5	Stack	114029			FALSE
GrupoJ	abc04	615	Stack	114029			FALSE
GrupoJ	abc06	0	Stack	119999			FALSE
GrupoJ	abc06	1	Stack	119999			FALSE
GrupoJ	abc06	2	Stack	119999			FALSE
GrupoJ	abc06	3	Stack	119999			FALSE
GrupoJ	abc06	290	Stack	119999			FALSE
GrupoJ	abc07	0	Stack	110696			FALSE
GrupoJ	abc07	1	Stack	110696			FALSE
GrupoJ	abc07	2	Stack	110696			FALSE
GrupoJ	abc07	3	Stack	110696			FALSE
GrupoJ	abc07	322	Stack	110696			FALSE
GrupoJ	ijk14	393	Stack	130070			FALSE
GrupoJ	ijk14	394	Stack	130070			FALSE
GrupoJ	ijk16	0	Stack	111081			FALSE
GrupoJ	ijk16	1	Stack	111081			FALSE

3.10.2 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.

3.10.3 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.

3.10.4 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.

3.10.5 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 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.11.2 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).

3.11.3 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).

3.11.4 E2

☒ Preliminar: arquivos devem estar na raiz

3.11.5 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 GrupoL

3.12.1 E5

- Não entregou.

3.12.2 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.

3.12.3 E3

☒ Veja os pontos não marcados como feitos abaixo, na E1; entendo

que eles ainda estão em aberto.

3.12.4 E2

- Nenhum comentário.

3.12.5 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 E5

☐ Usar a flag -I do compilador ao invés de incluir com caminhos

relativos tal como "`../include/iloc.h`".

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.13.2 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.

3.13.3 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.

3.13.4 E2

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

☒ Usar wildcards do makefile para simplificar as regras

3.13.5 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 GrupoN

3.14.1 E5

- Não entregou.

3.14.2 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.

3.14.3 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.

3.14.4 E2

☐ Melhorar o makefile para que se possa usufruir de um sistema de

compilação que permita compilação parcial dos vários fontes do projeto.

3.14.5 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 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.15.2 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.

3.15.3 E3

☒ Veja os pontos não marcados como feitos abaixo, na E1; entendo

que eles ainda estão em aberto.

3.15.4 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.

3.15.5 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 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.16.2 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.

3.16.3 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.

3.16.4 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.

3.16.5 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 E5

- Não submetido.

3.17.2 E4

- Não submetido

3.17.3 E3

- Não submetido

3.17.4 E2

- Não submetido

3.17.5 E1

- Não submetido

3.18 DONE GrupoR

3.18.1 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.18.2 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.

3.18.3 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).

3.18.4 E2

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

um número de linha

3.18.5 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 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.19.2 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.

3.19.3 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).

3.19.4 E2

☒ Nos comentários que categorizam as regras gramaticais, usar texto

não em uppercase total.

3.19.5 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 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.20.2 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.

3.20.3 E3

☒ Veja os pontos não marcados como feitos abaixo, na E2; entendo

que eles ainda estão em aberto.

3.20.4 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.

3.20.5 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 DONE GrupoZ

3.21.1 E5

- Sem comentários.

Group	Test	Value	Type	Address.cor	Address.mem	Dist	Correto

3.21.2 E4

- Não submetido.

3.21.3 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).

3.21.4 E2

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

- Realmente ficou faltando.

3.21.5 E1

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

4 Pesos

Grupo	E5.P
GrupoA	1
GrupoB	1
GrupoC	1
GrupoD	1
GrupoE	1
GrupoF	1
GrupoG	1
GrupoH	1
GrupoI	1
GrupoJ	1
GrupoK	1
GrupoM	1
GrupoO	1
GrupoP	1
GrupoR	1
GrupoS	1
GrupoT	1
GrupoZ	1

5 Final

Grupo	Etapa	E5.O	E5.S	E5.P
GrupoA	E5	10	10	1
GrupoB	E5	10	10	1
GrupoC	E5	10	10	1
GrupoD	E5	10	10	1
GrupoP	E5	10	10	1
GrupoS	E5	10	10	1
GrupoT	E5	10	10	1
GrupoZ	E5	10	10	1
GrupoH	E5	10	9.65	1
GrupoO	E5	10	9.65	1
GrupoE	E5	10	9.3	1
GrupoF	E5	10	9.3	1
GrupoG	E5	10	9.3	1
GrupoI	E5	10	9.3	1
GrupoK	E5	10	9.3	1
GrupoM	E5	10	9.3	1
GrupoR	E5	10	9.3	1
GrupoJ	E5	7.3	7.2	1

6 Recuperação

Grupos em recuperação da E5:

- **Apenas grupos que entregaram no prazo** conforme regramentos
- Em Moodle já configurado para tal, use o link "Recuperação E5"
- Política de recuperação ativada (vejam regras gerais)

Could not parse R result