Relatório do trabalho da disciplina de Processamento de Linguagens

Trabalho prático

Licenciatura em engenharia de Sistemas informáticos

Joaquim Cardoso - 10201

18 de dezembro de 2015

Índice

[Introdução 3](#_Toc439729891)

[Editor de imagem 4](#_Toc439729892)

[Bibliotecas 4](#_Toc439729893)

[Makefile 5](#_Toc439729894)

[Flex 6](#_Toc439729895)

[Vc.h 9](#_Toc439729896)

[Bison 11](#_Toc439729897)

[Porque usar “%error-verbose”? 11](#_Toc439729898)

[Editor de imagem com arvore sintática abstrata 15](#_Toc439729899)

[Vc.h 15](#_Toc439729900)

[Bison 17](#_Toc439729901)

[Intreperter 18](#_Toc439729902)

[Testes 19](#_Toc439729903)

[Conclusão 23](#_Toc439729904)

[Bibliografia 25](#_Toc439729905)

**Lista de Tabelas**

[Tabela 1 – Flag’s 5](#_Toc439727096)

**Lista de Figuras**

[Figura 1 - Fases de uma compilação 3](#_Toc439729906)

[Figura 2- Makefile 5](#_Toc439729907)

[Figura 3- Declarações em linguagem C 6](#_Toc439729908)

[Figura 4 – Definições 6](#_Toc439729909)

[Figura 5 - Regras 6](#_Toc439729910)

[Figura 6 - Regras continuação 7](#_Toc439729911)

[Figura 7 - Sub rotinas 8](#_Toc439729912)

[Figura 8 - Estruturas de dados 9](file:///C:\Users\Utilizador\Desktop\PL\Tp\Relatorio-PL%20Avançado.docx#_Toc439729913)

[Figura 9- Estruturas de dados continuação 9](file:///C:\Users\Utilizador\Desktop\PL\Tp\Relatorio-PL%20Avançado.docx#_Toc439729914)

[Figura 10 - Assinatura dos métodos 10](#_Toc439729915)

[Figura 11 - Assinatura dos métodos continuação 10](file:///C:\Users\Utilizador\Desktop\PL\Tp\Relatorio-PL%20Avançado.docx#_Toc439729916)

[Figura 12 - Bibliotecas utilizadas e declarações em C 11](#_Toc439729917)

[Figura 13 – Definições 11](#_Toc439729918)

[Figura 14 - Definições continuação 12](#_Toc439729919)

[Figura 15 - Regras 12](#_Toc439729920)

[Figura 16 - Regras continuação 13](file:///C:\Users\Utilizador\Desktop\PL\Tp\Relatorio-PL%20Avançado.docx#_Toc439729921)

[Figura 17 - Regras continuação 14](#_Toc439729922)

[Figura 18 – Sub-rotinas 14](#_Toc439729923)

[Figura 19 - Estrutura de dados 15](#_Toc439729924)

[Figura 20 – Assinatura dos métodos utilizados para a ast 16](#_Toc439729925)

[Figura 21 - Alteração ao nível do código c de cada produção 17](file:///C:\Users\Utilizador\Desktop\PL\Tp\Relatorio-PL%20Avançado.docx#_Toc439729926)

[Figura 22 - Alteração realizada para suporte de ast 17](file:///C:\Users\Utilizador\Desktop\PL\Tp\Relatorio-PL%20Avançado.docx#_Toc439729927)

[Figura 23 – Adição da estrutura de dados no union e type 17](file:///C:\Users\Utilizador\Desktop\PL\Tp\Relatorio-PL%20Avançado.docx#_Toc439729928)

[Figura 24 – Exerto de casos para o método executar 18](#_Toc439729929)

[Figura 25 - Desenho de vários retângulos preenchidos 19](#_Toc439729930)

[Figura 26 - Criação de uma nova imagem e alteração da cor predefinida 20](#_Toc439729931)

[Figura 27 - Utilização do comando "LINHAS" 20](#_Toc439729932)

[Figura 28 - Simulação de introdução de um ficheiro errado 21](#_Toc439729933)

[Figura 29 - Teste do comando "ABRIR" 21](#_Toc439729934)

[Figura 30 - Desenho de um circulo e um circulo preenchido 22](#_Toc439729935)

[Figura 31 - Demonstração de ciclos e variáveis 22](#_Toc439729936)

# Introdução

Este capítulo resume os principais temas de interesse na conceção do trabalho prático para a disciplina de processamento de linguagens, lecionada pelo Professor Doutor Óscar Ribeiro, no âmbito da aquisição de experiência sobre analisadores léxicos e sintáticos bem como a definição de ações semânticas, tal como a especificação de uma gramática concreta, construção de um reconhecedor léxico e sintático utilizando as ferramentas flex e bison e desenvolver um gerador de código associado às ações, foi proposto este trabalho prático, onde iremos aplicar os diversos conhecimentos transmitidos nas aulas, sendo que nenhum elemento do grupo teve noções ou contacto com processamento de linguagens antes de frequentar a UC para a qual se destina este trabalho.

Resumidamente, as três principais etapas e objetivos do projeto que contribuem para um melhoramento significativo do desempenho e velocidade, são:

A especificação da gramática concreta da linguagem de entrada;

A construção de um reconhecedor para a linguagem criada;

Desenvolver o gerador de código de forma a produzir uma reposta completamente positiva, de acordo com o enunciado do trabalho prático.



Figura 1 - Fases de uma compilação

# Editor de imagem

## Bibliotecas

**vc.h** é a nossa biblioteca que importa a estrutura de dados e dá a conhecer.

**Stdio.h** é uma biblioteca padrão da linguagem C. Possui definições relativas às operações de entrada/saída, tais como a leitura de dados a partir do teclado e a exibição de informações no ecrã.

**Ctype.h** é uma biblioteca da linguagem C que possui funções para classificar e transforma carateres individuais.

**String.h** fornece funções e definições da biblioteca padrão da linguagem C para a manipulação de arrays de carateres e regiões de memória.

**Stdlib.h** é uma biblioteca da linguagem C e possui funções que envolvem a alocação de memória, controlo de processos, conversões, pesquisa entre outras coisas.

**Stdint.h** é uma biblioteca da linguagem C que possui funções que declara um conjunto de tipos inteiros tal como o seu comprimento e correspondentes macros.

**Stdarg.h** é uma biblioteca da linguagem C que possui funções e um conjunto de macros que permite as funções aceitar listas de argumentos.

## Makefile

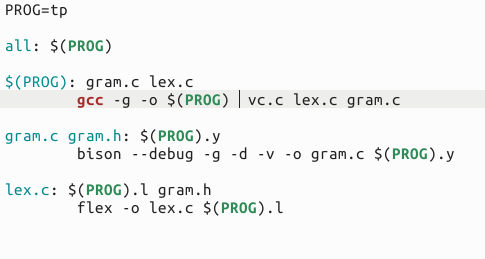


Figura 2- Makefile

|  |  |
| --- | --- |
| O que cada Flag faz no Makefile: | |
| GCC | |
| -g | Gera informações de debug para ser usado por GDB (debugger). |
| -o | Escreve a produção da compilação para um ficheiro de saída. |
|  | |
| Bison | |
| --debug | No ficheiro de implementação do analisador sintático define a macro YYDEBUG para 1 (caso este não esteja já definido), de modo a que seja possível a realização do debugging. |
| -g | Mostra uma representação gráfica do autómato do analisador calculado pelo bison, em formato .dot . |
| -d | Escreve um ficheiro de saída extra contendo definições de macros para os token, types e nomes definidos na gramática, tal como outras declarações, mas não aceita um ficheiro de argumento desde que POSIX Yacc exija que -d possa ser encaixado com outras opções mais pequenas. |
| -v | Escreve um ficheiro de saída adicional com descrições detalhadas da gramática e do analisador. |
| -o | Especifica o nome do ficheiro de saída do analisador. |
|  | |
| Flex | |
| -o | Especifica o nome do ficheiro de saída do analisador. |

Tabela 1 – Flag’s

## Flex

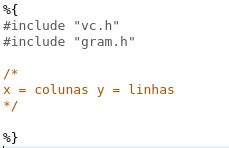


Figura 3- Declarações em linguagem C

Na figura 3, demonstra as bibliotecas utilizadas no flex. “vc.h” é uma biblioteca já referida anteriormente, já “gram.h” é a biblioteca gerada pelo bison para dar a conhecer os tipos de valores do yylval tal como os tokens utilizados. Na seguinte ilustração informamos o flex de algumas definições que iremos utilizar para realizar a análise léxica.

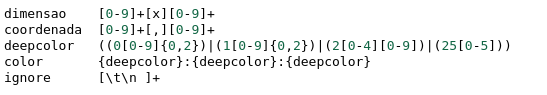


Figura 4 – Definições

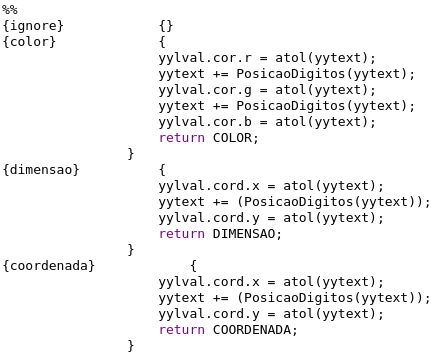


Figura 5 - Regras

Na figura 5 e 6 apresentamos todas das regras que irão ser aplicadas ao texto de entrada. Irão ser ignorados todos os espaços, tabulações e mudanças de linha, já quando aparecer uma cor em formato RGB (e.g. 255:255:255), vai retornar um token, mas antes vai retirar os dígitos de yytext e atribui-los ao yylval correspondente. A função atol serve para a conversão de uma “string” para um inteiro de 4 bytes, um long, a função atol irá converter todos os números que existirem até ao primeiro caráter não numérico, já a função strdup recebe uma “string” e retorna a mesma, no nosso trabalho foi utilizada para a atribuição ao yylval correspondente. O yytext é uma cópia do texto correspondente à regra lexical, existe duas regras bastante rígidas: para que o texto seja aceite tem que corresponder exatamente a uma das regras lexicais, para os efeitos da decisão da correspondência, será escolhida a regra lexical mais abrangente. O yylval é uma variável do bison que por defeito é inteiro, mas pode ter outros tipos de valores, esses tipos de valor têm que ser definidos no bison com um %union, o tipo de yylval é determinado por yystype. A função PosicaoDigito será explicada posteriormente.



Figura 6 - Regras continuação

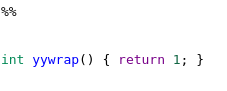


Figura 7 - Sub rotinas

Quando um analisador recebe uma indicação EOF, ele verifica a função yywrap. Ela é chamada pelo flex quando o texto de entra acaba, retorna 1 se acabou, 0 se irá ser preciso processar mais dados.

## Vc.h

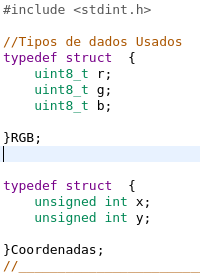
Neste subtítulo serão apresentados as estruturas de dados utilizadas e assinatura de métodos. Começando com a estrutura RGB da figura 8. RGB contém 3 variáveis do tipo uint8\_t, é um tipo de variáveis incomum, teoricamente funciona como um unsigned char, que suporta de valores de 0 até 255 tal como demonstra a tabela ASCII, sendo perfeito para o uso da atribuição de um valor para uma cor, pois o valor pode ser qualquer inteiro, mas o valor atribuído apenas será de 0 a 255, tal como o valor que será utilizado para definir uma cor. Estrutura denominada de “Coordenadas”, diferente do que como o próprio nome indica apenas irá suportar uma coordenada, o grupo considerou uma coordenada como um conjunto de dois valores inteiros sem sinal, logo foi declarado um inteiro sem sinal.

Figura - Estruturas de dados

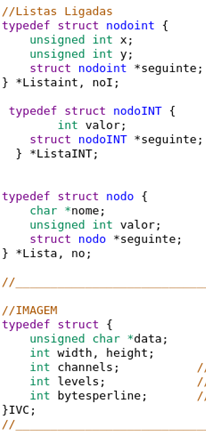


Figura - Estruturas de dados continuação

As seguintes três estruturas são listas ligadas, uma lista ligada é constituída por variáveis determinadas pelo programador e por uma variável especifica, uma variável do tipo apontador da própria estrutura. Logicamente na memória ocupada para um nó de uma lista ligada grande parte da memoria é ocupada por variáveis e o restante é para o apontador para uma próxima estrutura.

Neste trabalho prático foram utilizadas três listas ligadas diferentes, uma para que seja possível suportar várias coordenadas, outra que suporte um conjunto de valores, e por final foi desenvolvido uma lista que irá suportar variáveis e os seus valores.

Para concluir com as estruturas foi utilizada uma estrutura que suporte uma imagem de forma dinâmica. Esta estrutura contém os dados, que serão todos os píxeis que utilizarmos, a altura e a largura, os canais utilizados pela imagem e os bytes utilizados numa linha.

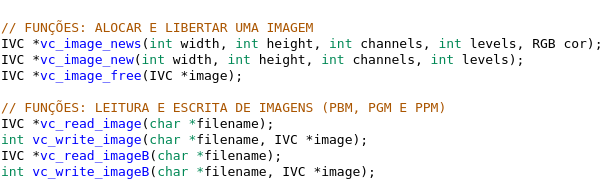


Figura 10 - Assinatura dos métodos

Na figura 10 e posteriormente na figura 11, são apresentados todos os métodos utilizados no trabalho prático. Na figura 10 são descritos os métodos referentes à imagem, alocar espaço dinamicamente, escrever uma imagem em ficheiro, ler uma imagem vinda de um ficheiro e libertar o espaço ocupado pela imagem na memória. Entretanto na figura 11 é apresentado diversos métodos. Métodos para pintar todo o tipo de desenhos pedidos no enunciado e ainda uma circunferência preenchida, existem também métodos para filtrar as coordenadas lidas, caso o utilizador se engane o programa não irá abortar, foi desenvolvido também métodos informativos para ajudar na correção dos erros que ocorreram no desenvolvimento do trabalho, tal como o uso de funções, adaptadas para a utilização no trabalho prático, utilizadas durante o atual semestre.

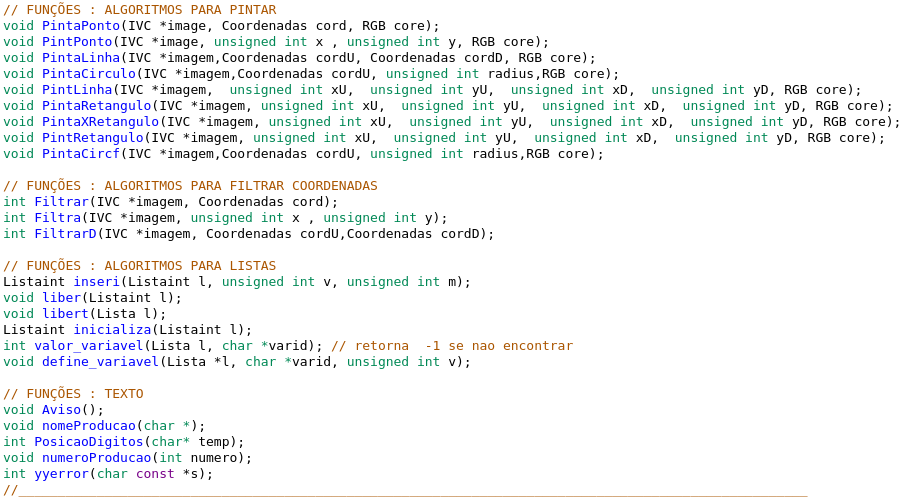


Figura - Assinatura dos métodos continuação

## Bison

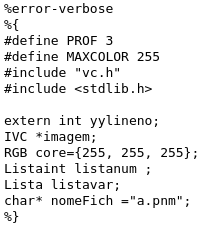


Figura 12 - Bibliotecas utilizadas e declarações em C

### Porque usar “%error-verbose”?

Quando ocorre um erro de sintaxe a ajuda para a correção do erro não ajuda muito, pois quando ocorre um erro de compilação o compilador dá uma ajuda sobre o que se passa e onde se passa, então foi utilizado o error-verbose. Esta definição para além de dizer que ocorreu um erro de sintaxe como está definido de padrão, o seu uso informa que foi enviado um determinado terminal que não era aguardado, e de seguida mostra qual o terminal esperado.

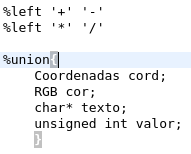


Figura 13 – Definições

Na figura 13, é apresentado os possíveis valores do yylval, sendo descrito pelo %union. É demonstrado também o uso de precedência.

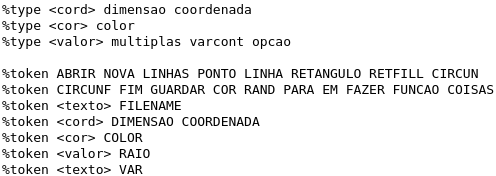


Figura 14 - Definições continuação

A figura 14 demonstra os terminais e os não terminais utilizado no trabalho, tal com o tipo de dados utilizado por eles. Para alguns dos terminais não foi determinado o tipo de dados pois não foi utilizado a passagem de valores por esse terminal, ou porque o terminal não necessitava de receber qualquer valor vindo flex.

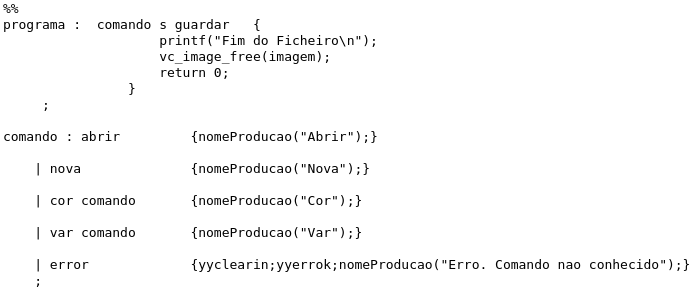


Figura 15 - Regras

A figura 15 dá a conhecer algumas das produções desenvolvidas para o trabalho, em conjunto com as figuras 16 e 17. Para ajuda do utilizador e do próprio desenvolvedor, foi necessário a apresentação de texto para identificar uma produção, sendo que foi desenvolvido um método capaz de mostrar dados e dar a perceber que percurso o texto realizou. Também apresentado nestas figuras, foi o uso de yyerrok, error e yyclearin, para a recuperação de erros.

Error é uma keyword do bison para identificar um erro na gramática, ou seja, quando o flex transporta um caráter ou um token que não é esperado pela produção no bison, o bison utiliza o error e o yyerrok, se o error estiver numa produção atingível, então será apresentado apenas uma mensagem de erro, se não estiver a utilizar o error na gramática o programa irá abortar com erro de sintaxe. Já o yyclearin atua de diferente forma, pois uma ação de recuperação de erro pode tomar para si a tarefa de encontrar o lugar correto para retomar a entrada. Neste caso, o lookahead anterior deve ser limpo.



Figura - Regras continuação

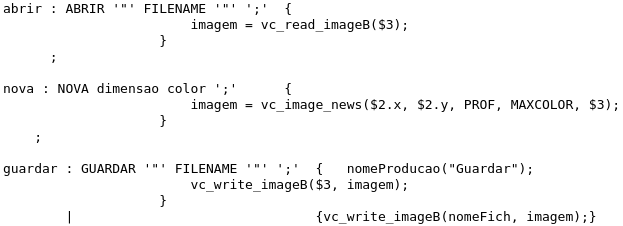


Figura 17 - Regras continuação

A estrutura de dados que suporta a imagem foi cedida pelo professor da unidade curricular de visão por computador em anos letivos anteriores. Foi então utilizada pelo grupo para o suporte de imagens de vários tipos, mais concretamente as imagens que utilizam um tipo de dados binário. Como é apresentado na figura 18, a utilização de métodos capazes de suportar uma imagem com qualquer dimensão e cor, sendo que os inteiros que fazem parte da dimensão sejam números inteiros positivos e que individualmente não excedam de 4 bytes de memória.

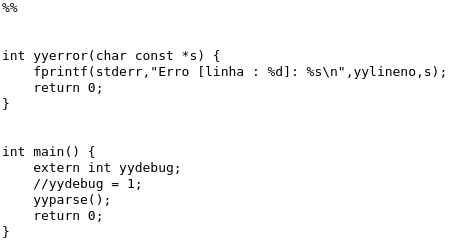


Figura 18 – Sub-rotinas

É chamada o método yyerror quando acontece por algum motivo um erro, então ao acontecer um erro é dito em que produção e por que motivo aconteceu o erro, sendo então yyerror é utilizado para a manipulação de erros. Outra curiosidade que foi encontrada na durante a fase de pesquisa foi o yydebug. O yydebug permite o compilador gerar um debugging das produções executadas, o que torna muito eficientemente a análise de erros.

# Editor de imagem com arvore sintática abstrata

Este capítulo do relatório apresenta o uso da árvore sintática abstrata para a realização de comandos dados por uma determinada. A estrutura do flex manteve-se, enquanto a estrutura do bison foi um pouco modificada. De seguida iremos apresentar as alterações realizadas e novos ficheiros criados.

O uso de árvores para realizar determinadas ações pedidas no enunciado deste trabalho é essencial, o seu uso é um pouco percetível, enquanto que a sua construção pode ser um pouco árdua. Para pequenos trabalhos não parece ser aconselhável, mas para algumas operações complexas é imprescindível. No caso deste trabalho os loops foram a razão para o uso das árvores, pois tal como a linguagem C# tem a keyword params para o envio de “ilimitados” parâmetros, em linguagem C é preciso criar uma estrutura que suporte diversos comandos, e que aja de forma recursiva para execução de cada comando enviado para a estrutura.

## Vc.h

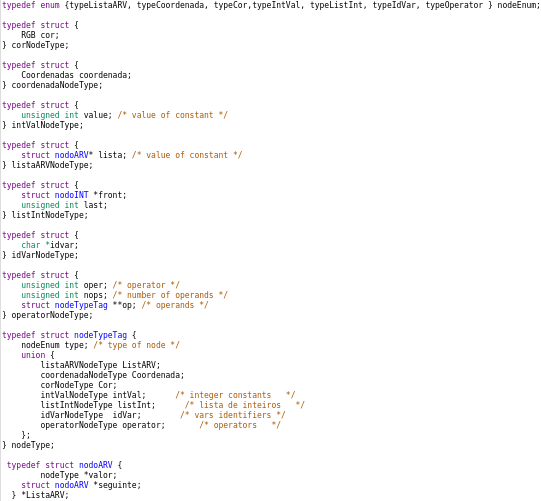


Figura 19 - Estrutura de dados

Na figura 19 é apresentada a estrutura da árvore abstrata, a base do código foi fornecido pelo professor da unidade curricular para o qual se destina este trabalho. Como se poderá observar na imagem em questão a arvore suporta variáveis, operações, coordenadas, e também dimensões pois o código é igual, lista de inteiros, lista de comandos, cor e inteiros.

Na seguinte ilustração é apresentado a assinatura dos métodos utilizados para a manipulação da árvore e suas operações.

De forma a consciencializar o presente código iremos fornecer uma breve explicação sobre os alguns dos seguintes métodos.

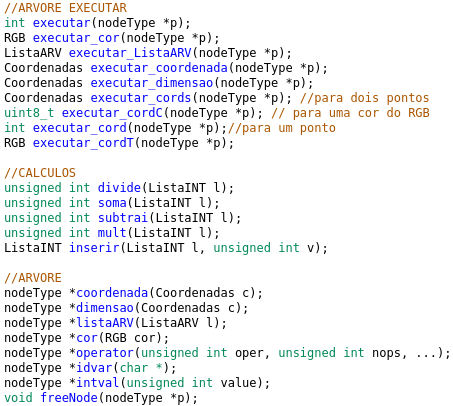


Figura 20 – Assinatura dos métodos utilizados para a ast

Métodos com a palavra executar correspondem à realização de um comando ou de um tipo de dados, pode ser também utilizado para a obtenção de valores de um comando para enviar de volta para a realização do próprio comando.

Métodos que correspondem a cálculos, como o próprio nome indica realiza operações aritméticas básicas e pode também servir para a adição de um valor a uma lista de valores.

Métodos que retornem um “nodeType\*” correspondem em métodos que foram utilizados para a obtenção de determinado tipo de dados ou estrutura de dados e a sua conversão para uma folha da árvore.

## Bison



Figura - Alteração ao nível do código c de cada produção

Figura - Alteração realizada para suporte de ast

No bison as alterações foram relativamente pequenas. A adição consistiu na utilização de métodos que recebem como parâmetro determinados comandos, e a sua limpeza após execução. Também passou pela adição do novo tipo de dados “nodeType” que corresponde a uma árvore abstrata, tal como a caracterização dos não terminais com a keyword type.

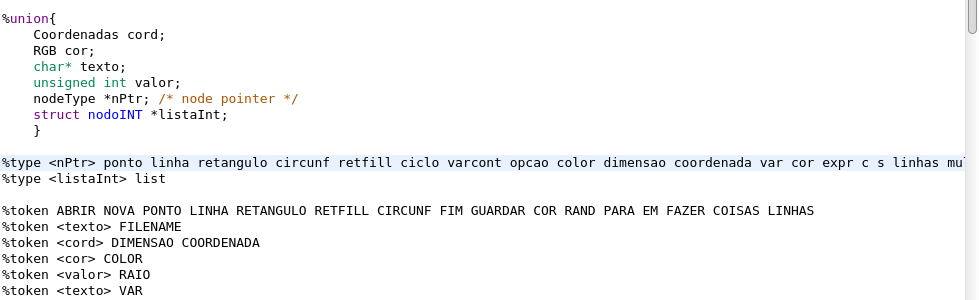


Figura – Adição da estrutura de dados no union e type

## Intreperter

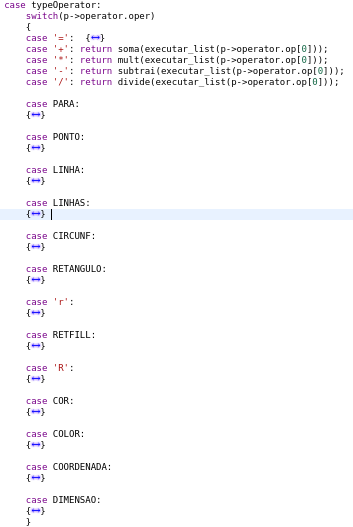


Figura 24 – Exerto de casos para o método executar

O ficheiro “intreperter” foi desenvolvido no âmbito da separação do código de execução da árvore com os demais métodos desenvolvidos anteriormente.

Este ficheiro está responsável por cada folha da árvore, tal como a própria árvore. Dentro do método executar como demonstra a figura 24 existem diversas operações suportadas para a execução de determinados comandos. Cada “case” tem um código desenvolvido especificamente para cada comando.

# Testes

Neste capítulo serão apresentados alguns textos de entrada, tal como a execução do trabalho prático e o seu determinado resultado. Durante a execução do ficheiro executável pode-se verificar todos os comandos recebidos pelos ficheiros de entrada, e ainda na mesma figura é possível verificar todo o conteúdo do texto de entrada. Para todos os testes realizados, presentes ou não no relatório, seram fornecidos tal como o código fonte.

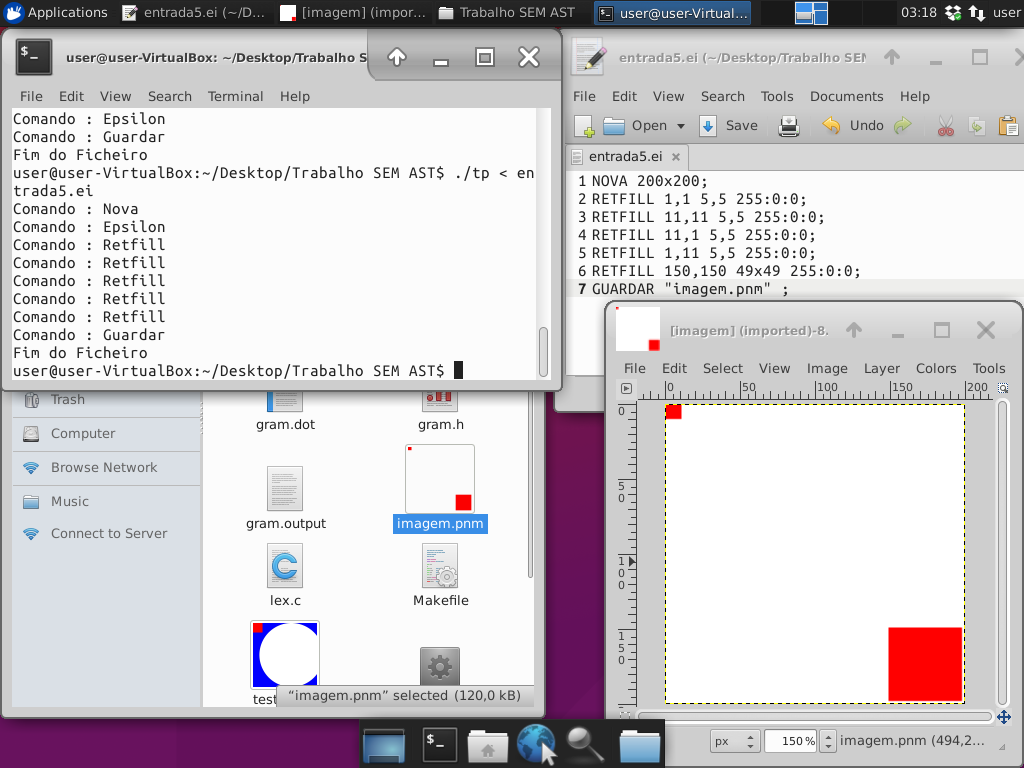


Figura 25 - Desenho de vários retângulos preenchidos

Na anterior figura foi realizado um teste para o desenho de um retângulo preenchido em todas as direções possíveis.

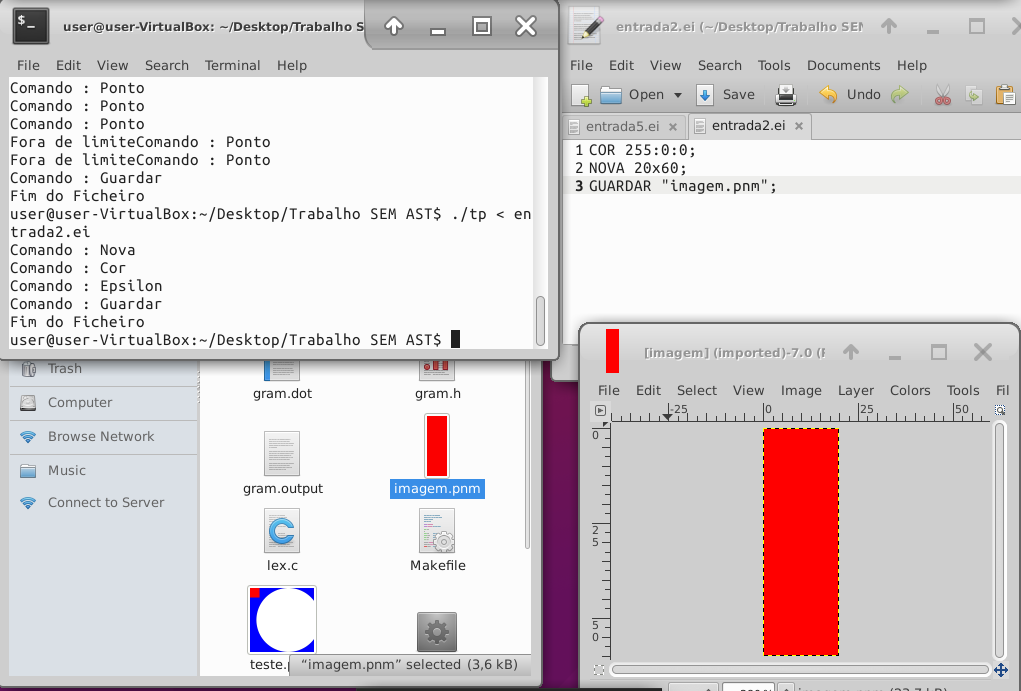


Figura 26 - Criação de uma nova imagem e alteração da cor predefinida

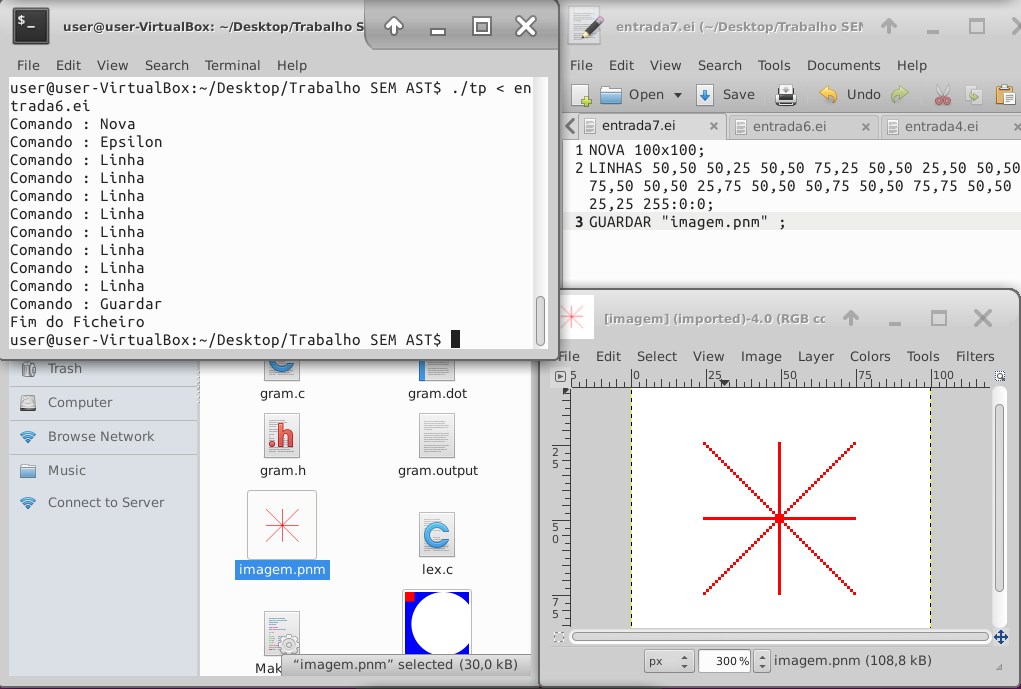


Figura 27 - Utilização do comando "LINHAS"

Na figura 27 foi realizado um teste para o desenho de um linhas em todas as direções possíveis.

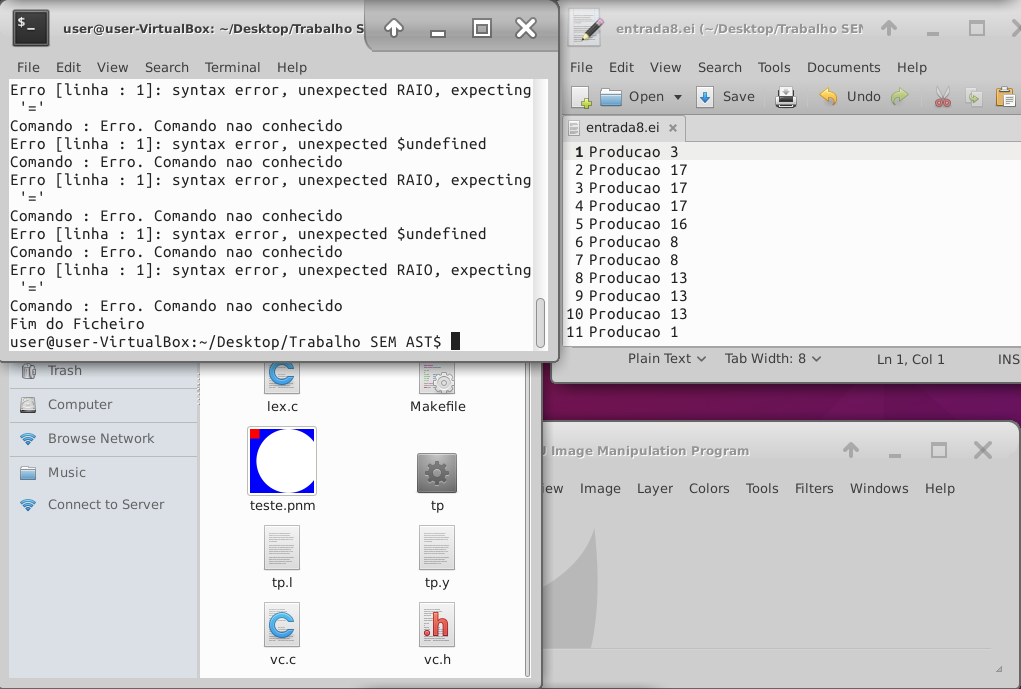


Figura 28 - Simulação de introdução de um ficheiro errado

Na figura 28 foi realizado um de verificação de comportamento do programa quando recebe o input errado.

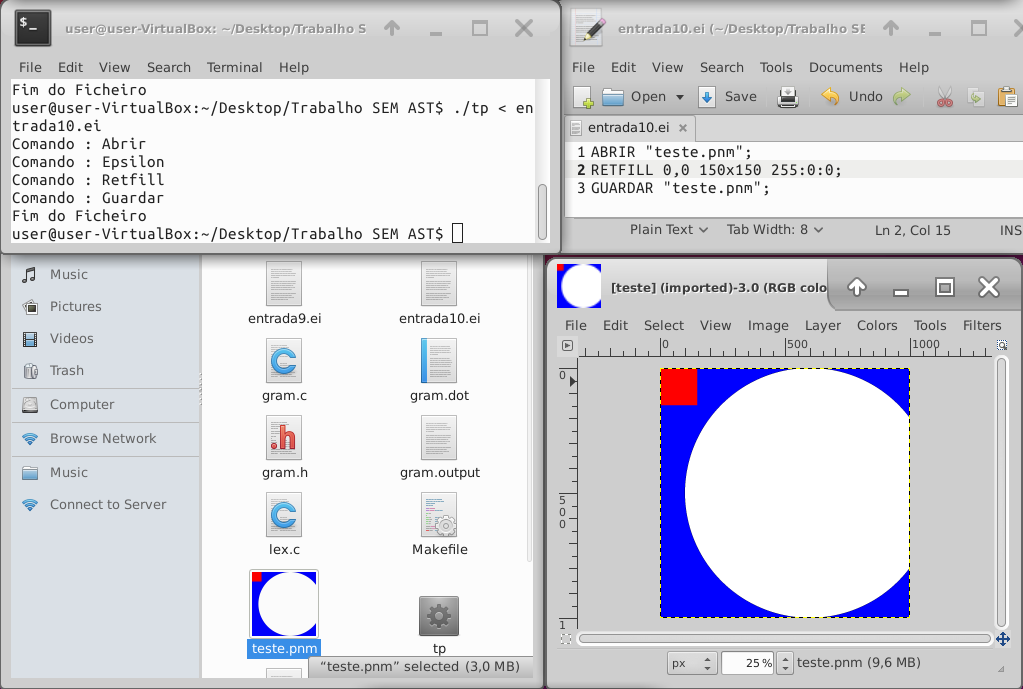


Figura 29 - Teste do comando "ABRIR"

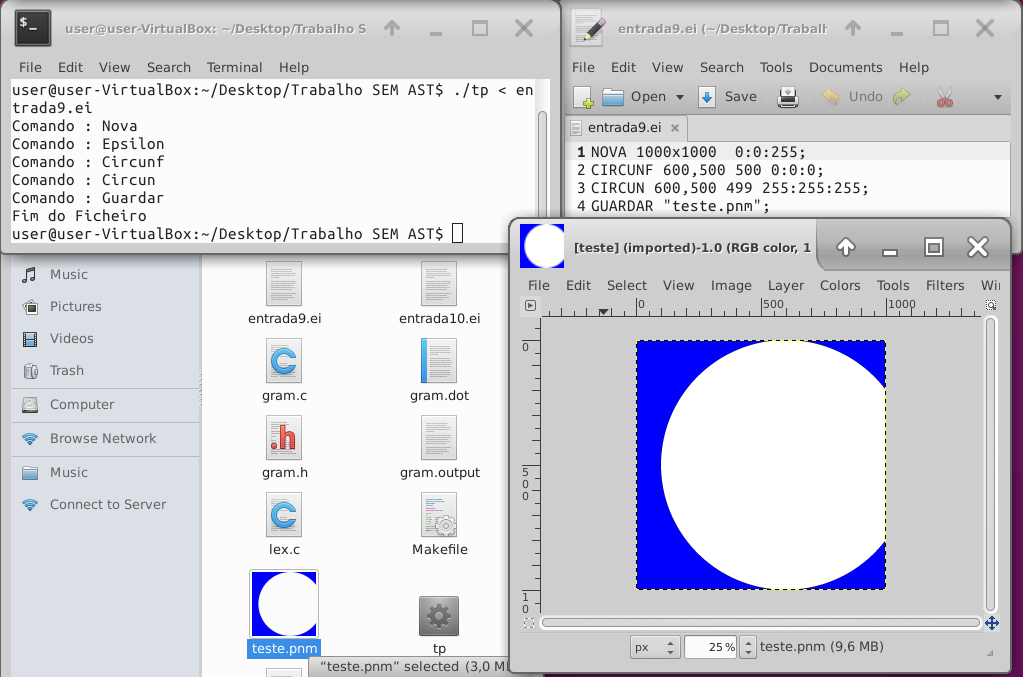


Figura 30 - Desenho de um circulo e um circulo preenchido

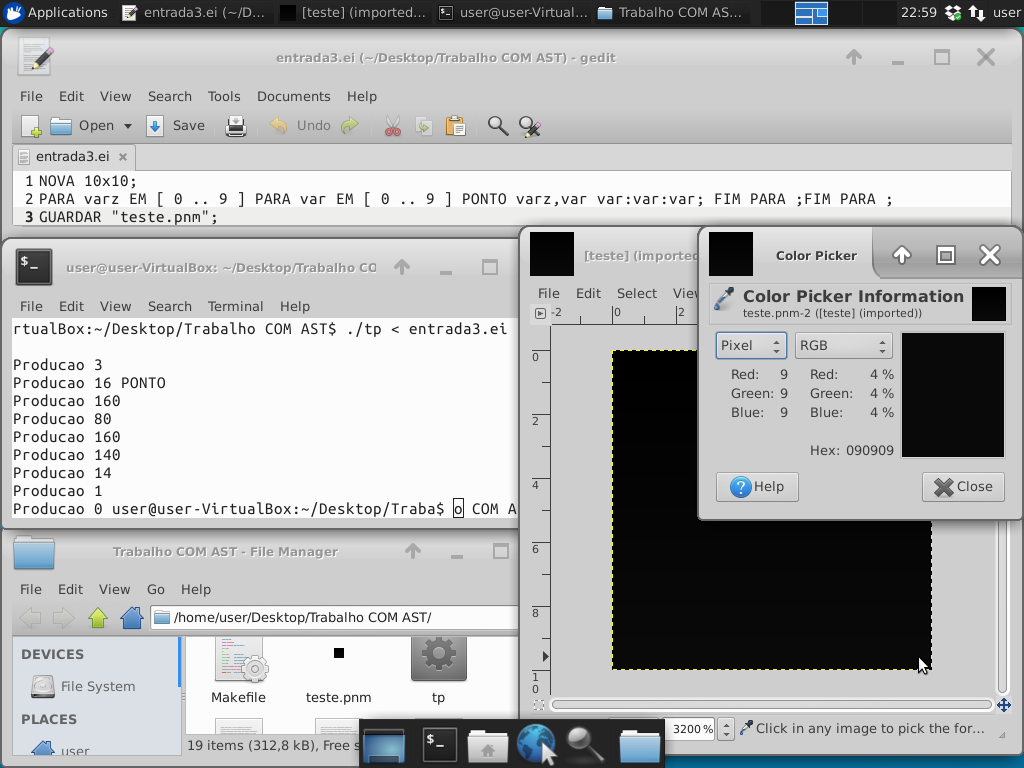


Figura 31 - Demonstração de ciclos e variáveis

Na anterior figura foi realizado um teste para o desenho pontos utilizando árvores abstratas e ciclos dentro de ciclos.

# Conclusão

A conceção e desenvolvimento de um trabalho, capaz de oferecer ao utilizador uma diversidade de opções e avisos prévios em cada erro dado, exigiram a execução de um conjunto de etapas. O trabalho desenvolvido foi dividido em duas partes principais, um referente á pesquisa de informação, outra parte correspondente à construção de analisadores léxicos e sintáticos utilizando respetivamente as tecnologias flex e bison. Uma boa programação é fundamental para prevenir a existência de erros grotescos.

O desenvolvimento do analisador léxico e sintático foi construído com sucesso. Até se obter um programa funcional em linguagem em texto estruturado e que cumprisse com os requisitos iniciais propostos no enunciado, realizou-se diversos testes ao código implementado.

A construção do compilador para linguagem em texto estruturado passou por três etapas principais na sua implementação, nomeadamente três geradores de análise:

1. Gerador de Análise Léxica (Scanner);
2. Gerador de Análise Sintática e semântica (*Parser*);
3. Interpretador.

Através da análise léxica do código implementado (função do flex) foi possível identificar e reconhecer cada símbolo característico (*token*) da linguagem (e.g. PONTO, COORDENADA, variáveis, nomes de ficheiros, operadores, etc.).

A análise sintática e semântica (ação do bison) permitiu criar uma estrutura hierárquica resultantes das regras gramaticais e de produção que cada *token* enviado da análise léxica formou, quando conjugados com outros *tokens*. O significado semântico das regras de produção geradas foi interpretado, e através de um conjunto de funções realizadas em linguagem C, foram definidas as respetivas instruções a serem executadas consoante esse mesmo significado e objetivo.

É possível assim, projetar e implementar um analisador léxico, uma vez que, as expressões regulares são conhecidas.

Em suma, podemos apresentar as principais conclusões retiradas com a realização do trabalho:

1) Simulador de linguagem em texto estruturado;

Testes realizados na linguagem, comprovam a eficiência das definições gramaticais produzidas na construção do compilador flex e bison.

2)Estrutura gramatical hierarquizada definida e pronta a ser utilizada para geração de código.

O interpretador foi desenvolvido com sucesso e também implementado pois este mesmo chegou a atingir os objetivos pretendidos pelo grupo. As dificuldades na conceção deste trabalho estão ligadas á falta de conhecimento de elaboração de imagens, tal como algoritmos de desenho, arquitetura lógica para suporte de uma imagem e por final falta de conhecimento de estruturas de dados interligadas com a linguagem de programação C, que provou ser bastante árdua para a conclusão dos objetivos pretendidos, sendo que devido à falta de familiaridade fora do âmbito da disciplina foi o principal fator que foi imposto como barreira. Com dedicação e esforço todas dificuldades foram todas ultrapassadas, umas mais facilmente que outras, o grupo apresenta-se e orgulhosos devido a todas as barreiras e obstáculos ultrapassados. Sendo que toda a falta de conhecimento foi um incentivo para procurar soluções e seguir em frente. Podemos assim afirmar que o trabalho foi concluído e imensamente interessante.

O grupo expressa o seu profundo e sincero agradecimento para com o Professor Doutor Óscar Ribeiro pela sua inteira disponibilidade, compreensão e transmissão de conhecimento durante o semestre, tal como o apoio e opiniões fundamentais para a conceção do trabalho prático.

# Bibliografia

[1] Engenharia de Linguagens (2011/2012)

<http://twiki.di.uminho.pt/twiki/bin/view/Education/EL/WebHome> [consultado. Dez. 2015].

[2] Principles of Compilers

<http://twiki.di.uminho.pt/twiki/pub/Education/EL/WebHome/Principles_of_Compilers__A_New_Approach_to_Compilers_Including_the_Algebraic_Method.pdf> [consultado. Dez. 2015].

[3] Parse Tree Listeners

<https://github.com/antlr/antlr4/blob/master/doc/listeners.md> [consultado. Dez. 2015].

[4] Processamento Morfológico e Correção Ortográfica do Português

<http://www.linguateca.pt/Repositorio/TeseMestradoJoseCarlosMedeiros.pdf> [consultado. Dez. 2015].

[5] Processamento de Linguagem Natural para Produtos de Seguros

<https://repositorio-aberto.up.pt/bitstream/10216/74373/2/99783.pdf> [consultado. Dez. 2015].

[6] [Model of access to natural language sources in electronic government; Modelo de acesso a fontes em linguagem natural no governo eletrónico](http://biblioteca.versila.com/2768657/model-of-access-to-natural-language-sources-in-electronic-government-modelo-de-acesso-a-fontes-em-linguagem-natural-no-governo-electronico)

<http://biblioteca.versila.com/2768657/model-of-access-to-natural-language-sources-in-electronic-government-modelo-de-acesso-a-fontes-em-linguagem-natural-no-governo-electronico> [consultado. Dez. 2015].

[7] Aquamentus Flex e Bison

<http://aquamentus.com/flex_bison.html> [consultado. Dez. 2015].

[8] Inescid Lisboa

<https://fenix.tecnico.ulisboa.pt/downloadFile/3779580668135/Estagios_Verao_MEEC_inescid.pdf> [consultado. Dez. 2015].

[9] flex & bison (O’REILLY)

<http://web.iitd.ac.in/~sumeet/flex__bison.pdf> [consultado. Dez. 2015].

[10] Utilização de Sistemas Móveis para Atualização de dados Georrefenciados

<https://repositorio-aberto.up.pt/bitstream/10216/12668/2/Texto%20integral.pdf> [consultado. Dez. 2015].

[11] Simulador de Linguagem em Texto Estruturado para Autómato TSX3721

<https://www.google.pt/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&ved=0ahUKEwid94G25PHJAhVCTBoKHYmXB4sQFgg4MAQ&url=http%3A%2F%2Frun.unl.pt%2Fbitstream%2F10362%2F14217%2F1%2FPecorelli_2014.pdf&usg=AFQjCNHhRHcjinC32X5GvoMTvLO4TYjT4Q&sig2=6Zqv840x5_vdwRJhwxszHQ&bvm=bv.110151844,d.ZWU&cad=rja> [consultado. Dez. 2015].

[12] Lexical Analysis With Flex, for Flex 2.5.37

<http://flex.sourceforge.net/manual/> [consultado. Dez. 2015].

[13] Gerador Automático de Formulários Dinâmicos

<https://www.google.pt/url?sa=t&rct=j&q=&esrc=s&source=web&cd=7&ved=0ahUKEwid94G25PHJAhVCTBoKHYmXB4sQFghGMAY&url=http%3A%2F%2Ffiles.alexanderthepoet.webnode.pt%2F200000067-5901759f9a%2FDISSERTA%25C3%2587%25C3%2582O%2520Gerador%2520autom%25C3%25A1tic23AbrilVFINAL.pdf&usg=AFQjCNEOWc8HSRXCkxmA3sVdmeJoUf2dpA&sig2=Et5NA6CTXdmOvFyU4SOWVQ&bvm=bv.110151844,d.ZWU&cad=rja> [consultado. Dez. 2015].

[14] GNU Operating System

<https://www.gnu.org/software/> [consultado. Dez. 2015].

[15] Introdução ao Flex <http://www.di.ubi.pt/~cbarrico/Disciplinas/Compiladores/Downloads/IntroducaoFlex.pdf> [consultado. Dez. 2015].

[16] A Compact guide to Lex & YACC by Tom Niemann <http://necc.di.uminho.pt/wiki/lib/exe/fetch.php?media=ap:2ano:plc:ap_plc_teoria_lex_yacc.pdf> [consultado. Dez. 2015].

[17] Bison

<http://dinosaur.compilertools.net/bison/> [consultado. Dez. 2015].

[18] YACC

<https://courses.cs.washington.edu/courses/cse322/10au/slides/10lexyacc-4up.pdf> [consultado. Dez. 2015].

[19] Structured Text (ST)

<http://www.kronotech.com/ST/introduction.htm>

[consultado. Dez. 2015].

[20] COMPILERS AND INTERPRETERS

<http://flint.cs.yale.edu/cs421/lectureNotes/c04.pdf>

[consultado. Dez. 2015].

[21] FORMAL LANGUAGES AND COMPILERS

<http://brasil.cel.agh.edu.pl/~11sustrojny/en/compiler/>

[consultado. Dez. 2015].

[22] Using Bison

<http://rosemary.umw.edu/~finlayson/class/fall13/cpsc401/notes/08-bison.html>

[consultado. Dez. 2015].

[23] Processamento de Linguagens (Capítulo 3 – Análise Sintática Parte 1)

<http://elearning.ipca.pt/1516/pluginfile.php/223071/mod_resource/content/4/ESI_2015-16_Processamento%20Linguagens-sebenta_cap%C3%ADtulo-3_parte1.pdf>

[consultado. Dez. 2015].

[24] Processamento de Linguagens (Capítulo 3 – Análise Sintática Parte 2)

<http://elearning.ipca.pt/1516/pluginfile.php/223397/mod_resource/content/2/ESI_2015-16_Processamento%20Linguagens-sebenta_cap%C3%ADtulo-3_parte2.pdf>

[consultado. Dez. 2015].

[25] Processamento de Linguagens (Capítulo 3 – Análise Sintática Parte 3)

<http://elearning.ipca.pt/1516/pluginfile.php/223665/mod_resource/content/1/ESI_2015-16_Processamento%20Linguagens-sebenta_cap%C3%ADtulo-3_parte3.pdf>

[consultado. Dez. 2015].

[26] Makefiles

<https://pt.wikibooks.org/wiki/Programar_em_C/Makefiles>

[consultado. Dez. 2015].

[27] Importância da Makefile

<https://aulasdec.wordpress.com/2011/07/28/utilitarios-make-makefiles-e-sua-importancia/>

[consultado. Dez. 2015].

[28] Makefile

<http://wiki.portugal-a-programar.pt/dev_geral:c:c_e_o_makefile>

[consultado. Dez. 2015].

[29] Bitmap/Bresenham's line algorithm

<http://rosettacode.org/wiki/Bitmap/Bresenham's_line_algorithm>

[consultado. Dez. 2015].

[30] Midpoint circle algorithm

<http://rosettacode.org/wiki/Bitmap/Midpoint_circle_algorithm#C>

[consultado. Dez. 2015].

[31] Código de estrutura de dados para uma imagem

[<http://elearning.ipca.pt/1516/my/>](http://rosettacode.org/wiki/Bitmap/Midpoint_circle_algorithm#C)[consultado. Dez. 2015].