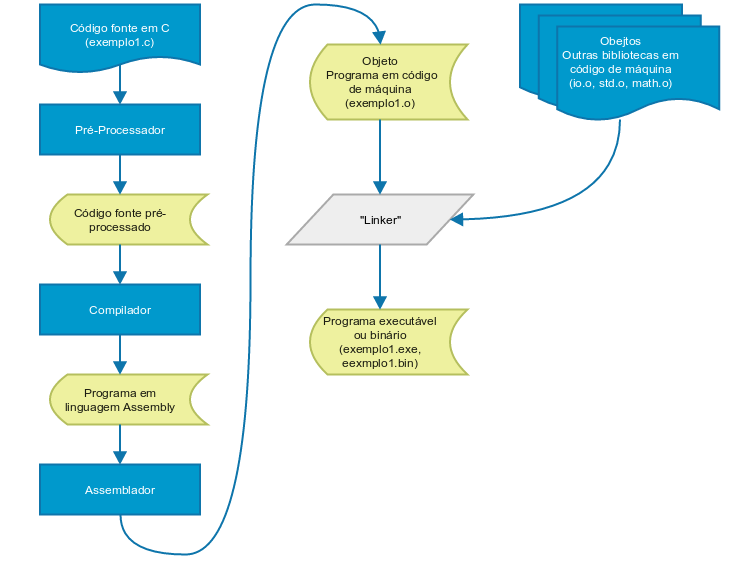
**Relatório 2 de Sistemas Digitais para a Mecatrônica/Sistemas Embarcados II**

**Aluno: Glênio Simião Ramalho N°: 11611EMT008**

**Questão 01) Liste e descreva o que são as 4 etapas do processo de compilação.**

As quatro etapas estão representadas no fluxograma abaixo:



Onde temos na ordem,

1. Pré-processamento:

Responsável por modificar o código-fonte do programa criado para corrigir alguns caracteres desnecessários, como espaços a mais, substituir macros (definidos pela #define MACRO) e incluir outro códigos (através das diretivas de pré-processamento). Essa etapa gera um código chamado de unidade de compilação.

1. Compilação:

Nessa etapa o compilador faz a análise sintática e semântica da unidade de compilação gerada na etapa precedente e gera então um código assembly correspondente.

1. Montagem:

Nessa etapa os comandos assembly são transformados em linguagem de máquina gerando o código-objeto do programa.

1. Ligação:

Etapa final do processo, onde todos os códigos-objeto do programa são combinados para criar finalmente o código executável.

**Questão 02) Desenvolva uma aplicação simples que demonstre o uso de múltiplos arquivos para a construção de uma aplicação em C.**

A minha aplicação está na pasta “Semana02/Carro” e é composta pelos arquivos “carros.c”, “bateria.c”, “bateria.h”, “tanque.c” e tanque.h”. Ela concerne simplesmente em mostrar inicialmente os estados da bateria do carro (em %) e do tanque de gasolina (em L) e calcular os estados dos mesmo após o uso do som do carro em h e da quantidade de km rodados, printando no final os valores correntes da bateria e do tanque de combustível.

Une image contenant texte

Description générée automatiquement

Compilando o código

Une image contenant texte

Description générée automatiquement

Rodando o código

Nesse exemplo, eu criei dois códigos representando a bateria e o tanque de combustível do carro para exemplificar como eles são chamados pelo código principal “carro.c”, de forma a representar bem simplificadamente componentes distintos de um carro, mas o uso de multiarquivos em C pode ajudar com estruturas bem complexas, como um supermercado por exemplo, onde temos diversos componentes como o caixa, o estoque, funcionários etc.

Une image contenant texte

Description générée automatiquement

carro.c declarando as bibliotecas bateria.h e tanque.h

**Questão 03) O compilador gcc permite fornecer parâmetros extras, que modificam desde a emissão de erros até o binário final, o otimizando para determinados comportamentos. Explique a função e crie um exemplo para demonstrar a funcionalidade dos seguintes parâmetros:**

**a) -static**

A opção -static cria um executável contendo todo o código necessário para a execução do código compilado, ou seja, o código não vai precisar procurar uma determinada função utilizada de uma outra biblioteca do sistema fora do próprio código, pois ele já vai ter o código dessa função dentro dele.

Um exemplo é um código simples com apenas um printf(), esta função faz parte da biblioteca stdio.h, se usarmos a opção -static no momento da compilação, o código final terá todo o código que faz a função printf() funcionar dentro dele mesmo, não precisando mais buscar a função dentro da biblioteca externa.

Essa opção, no entanto, torna o código compilado muito mais pesado, visto que todos os códgos das funções externas estão presentes no código final, aumentando-o de tamanho.

Exemplo:

|  |
| --- |
| #include<stdio.h>  int main()  {  printf("Hello\n");  } |

Código static.c

Compilando o código com e sem a opção -static e mostrando a diferença de tamanho:

Une image contenant texte

Description générée automatiquement

Podemos ver que que o código compilado com a opção -static (*static)* tem um tamanho de 871760 enquanto que o código sem essa opção (*no\_static*) possui um tamanho muito menor de 16696.

**b) -g**

Usado para gerar informações de “debug” do programa, usado pelo *GDB debugger.*

A opção -g é um dos níveis de *debug*, os quais são:

|  |  |
| --- | --- |
| **option** | **description** |
| -g0 | Sem informação de debug |
| -g1 | Informação mínima |
| -g | Informação de debug padrão |
| -g3 | Informação máxima |

Exemplo:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main()  {  int a = 5, b = 2;    printf("b vale %d", b);  } |

Código g.c

Une image contenant texte

Description générée automatiquement

Compilação e GDB

**c) -pedantic**

Verifica se o código está de acordo com as normas do padrão AINSI/ISO C, se o código não estiver ele não é rodado.

Exemplo:

|  |
| --- |
| #include <stdio.h>  int main()  {  double double b = 2l;  fprintf(stdout, "This is a non-conforming C program\n");    return 0;  } |

Código retirado do livro *“Kurt Wall. Linux Programming Unleashed.SAMS, 2007”*

|  |
| --- |
| $ gcc -pedantic pedant.c -o pedant  pedant.c: In function `main’  : pedant.c:9: warning: ANSI C does not support `long long’ |

Erro de compilação. retirado do livro *“Kurt Wall. Linux Programming Unleashed.SAMS, 2007”*

**d) -Wall**

Mostra na tela todos os warnings existentes no programa.

Exemplo:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main()  {  int a = 5, b = 2;    printf("b vale %d", b);  } |

Código wall.c

Une image contenant texte

Description générée automatiquement

Compilação

**e) -Os**

flag usada no gcc para otimizar a compilação de um código, existem vários níveis de otimização, cada um tendo um diferencial, no caso do -Os ele otimiza o tamanho do código, mas aumenta o tempo de compilação.

Abaixo uma tabela com as características do -Os:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Opção** | **Tempo de execução** | **Tamanho do código** | **Uso de memória** | **Tempo de compilação** |
| -Os | Reduz bastante |  |  | Aumenta bastante |



Exemplo de compilação

**f) -O3**

Assim como o -Os, -O3 também é um nível de otimização, sendo este mais focado na diminuição do tempo de execução, porém com aumento do uso de memória e do tempo de compilação.

Abaixo uma tabela com as características do -Os:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Opção** | **Tempo de execução** | **Tamanho do código** | **Uso de memória** | **Tempo de compilação** |
| -O3 | Reduz muito |  | Aumenta | Aumenta muito |



Exemplo de compilação

**Questão 04) Veja os cinco primeiros vídeos da seguinte lista:** [**https://www.youtube.com/watch?v=hrPxwKtedCc&list=PL3ZslI15yo2pCf0WpZmV-ga02kMPxKH3p&index=1**](https://www.youtube.com/watch?v=hrPxwKtedCc&list=PL3ZslI15yo2pCf0WpZmV-ga02kMPxKH3p&index=1)

Todos foram vistos

**FONTES:**

<https://www.rapidtables.com/code/linux/gcc/gcc-o.html#optimization>

<https://www.rapidtables.com/code/linux/gcc/gcc-g.html>

<https://www.youtube.com/watch?v=-vp9cFQCQCo&ab_channel=DaveXiang>

<http://www.inf.ufes.br/~pdcosta/ensino/2017-1-estruturas-de-dados/material/GuiaRapido_EDI.pdf>

<https://wiki.sj.ifsc.edu.br/index.php/AULA_5_-_Programa%C3%A7%C3%A3o_1_-_Gradua%C3%A7%C3%A3o>

<http://ulysseso.com/bonus/Docs/ProgMultiArquivosGcc.pdf>

<https://silo.tips/download/3-compilaao-e-estrutura-basica-de-um-programa-em-c>