Programação Orientada a Objetos para Redes de Computadores

Prof. Miguel Elias Mitre Campista

http://www.gta.ufrj.br/~miguel

PARTE 1

Conceitos Básicos de programação

O que é um Programa Computacional?

O que é um Programa Computacional?

- É um algoritmo expresso em uma **linguagem de programação** formal onde se conhece...
  - A maneira como representá-lo na linguagem de programação
  - A estrutura de representação dos dados
  - Os tipos e as operações suportados pelo computador

Como um Programa é Executado?

- Linguagens de programação
  - São projetadas em função da **facilidade na construção do código** e da **confiabilidade dos programas**
    - Quanto mais próximo a linguagem de programação estiver da forma de raciocínio humano, mais intuitivo se torna o programa e mais simples é a programação

```
#include <stdio.h>
main() {
    ENQUANTO condição satisfeita FAÇA
            execute ação 1;
    FIM DO ENQUANTO
    imprimir "Acabou";
}
```

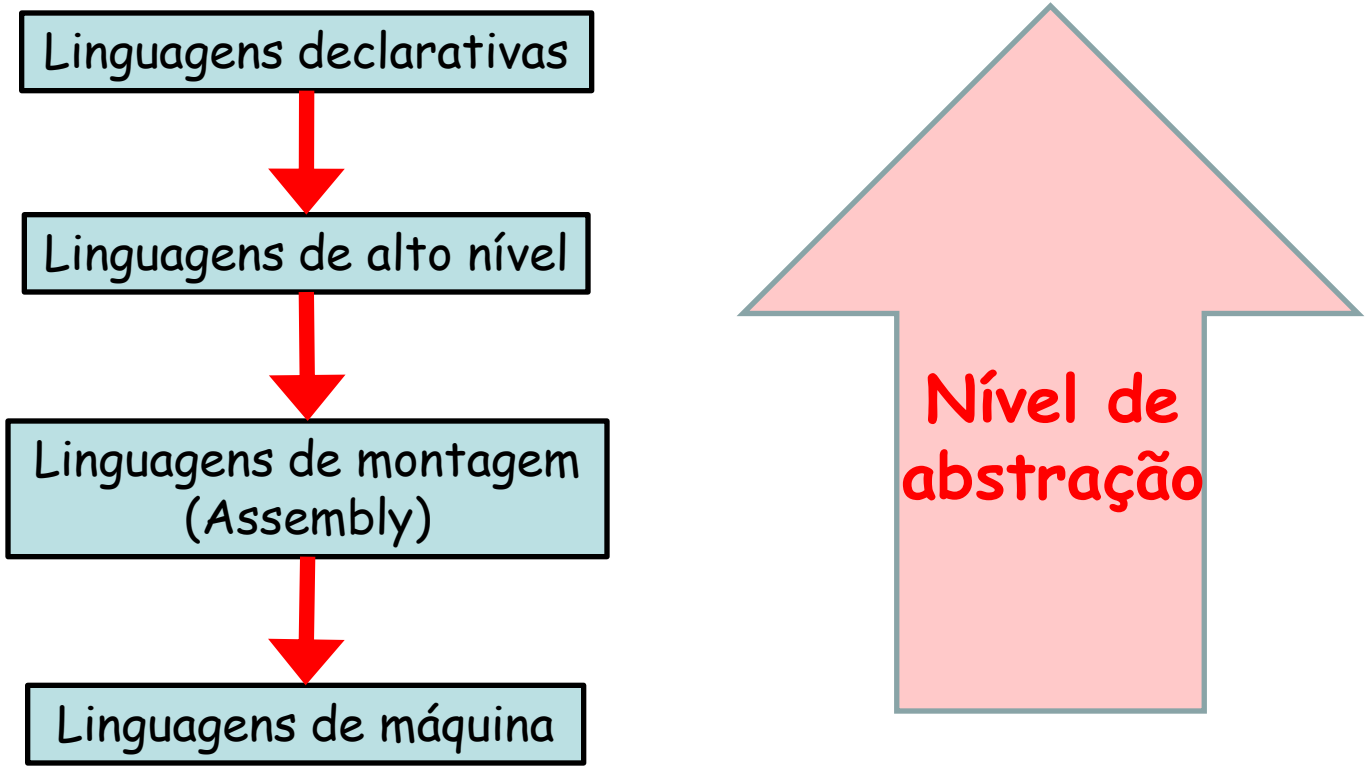
Como um Programa é Executado?

- Entretanto, computadores não entendem a linguagem humana...
  - Computadores entendem sequências de 0's e 1's
    - Chamada de linguagem de máquina

```
#include <stdio.h>
main() {
    ENQUANTO condição satisfeita FAÇA
            execute ação 1;
    FIM DO ENQUANTO
    imprimir "Acabou";
}
```

1	0	1	1	0
0	0	1	1	0
		...		
0	1	0	1	0
0	1	0	0	1

Arquitetura de Computadores em Multiníveis



Arquitetura de Computadores em Multiníveis

- Linguagens declarativas
  - Linguagens expressivas como a linguagem oral
    - Expressam o que fazer ao invés de como fazer
- Linguagens de nível alto
  - Abstraem do programador detalhes da arquitetura do computador para facilitar a leitura e a compreensão
    - Processador, memória, dispositivos de entrada e saída etc.
  - Possuem características de portabilidade já que podem ser transferidas de uma máquina para outra
  - São as linguagens típicas de programação
    - Exemplos: C++, C, Java, Fortran etc.

Arquitetura de Computadores em Multiníveis

- Linguagens de montagem e linguagens de máquina
  - Linguagens que dependem da arquitetura da máquina
    - Linguagem de montagem é uma representação simbólica da linguagem de máquina associada

- Linguagem de máquina
  - Linguagem de nível baixo
    - Depende da arquitetura do processador
      - Cada processador define uma arquitetura de conjunto de instruções (*Instruction Set Architecture* - ISA)

Arquitetura de Computadores em Multiníveis

- Existem duas maneiras para decodificar programas
  - Programa em linguagem de nível alto para programa em linguagem de nível baixo
    - **Interpretação**
    - **Tradução**

Interpretação

- Na interpretação **cada** comando em linguagem de programação de alto nível é decodificado e executado
  - Processo realizado **durante** a execução do programa
    - Um comando por vez

- Para isso,
  - Há um programa interpretador sendo executado
    - Um interpretador para cada arquitetura de processador
  - Cada comando do código é visto por esse interpretador como um dado de entrada

Interpretação

- O computador executa programas auxiliares escritos em linguagem de máquina para interpretar cada comando do programa
  - Os programas auxiliares são invocados em uma ordem apropriada de acordo com a ordem de execução do programa
- Etapas da interpretação
  - Obter o próximo comando
  - Examinar e decodificar o comando
  - Executar as ações

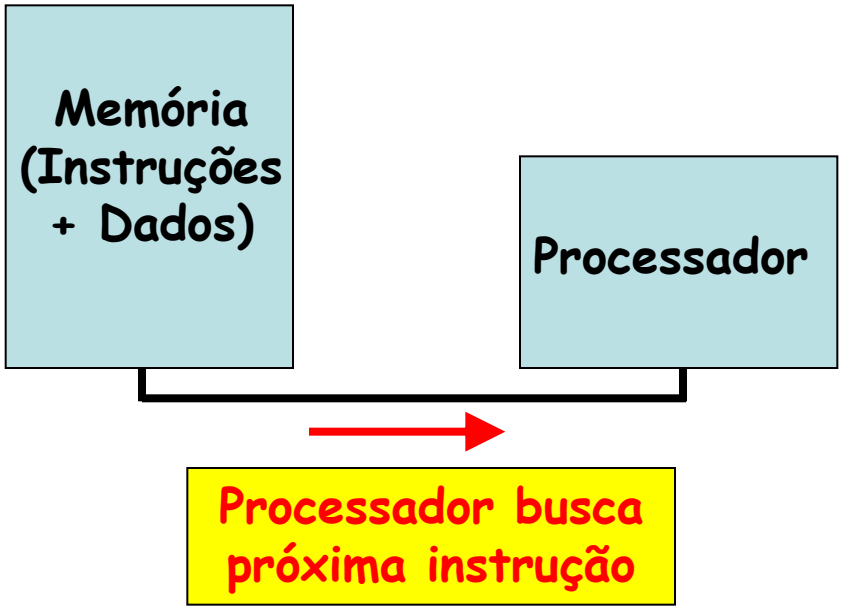
Interpretação

- O computador executa programas auxiliares escritos em linguagem de máquina para interpretar cada comando do programa
  - Os programas auxiliares são invocados em uma ordem apropriada de acordo com a ordem de execução do programa
- Etapas da interpretação
  - Obter o próximo comando
  - Examinar e decodificar o comando
  - Executar as ações

Ciclo de execução de um programa

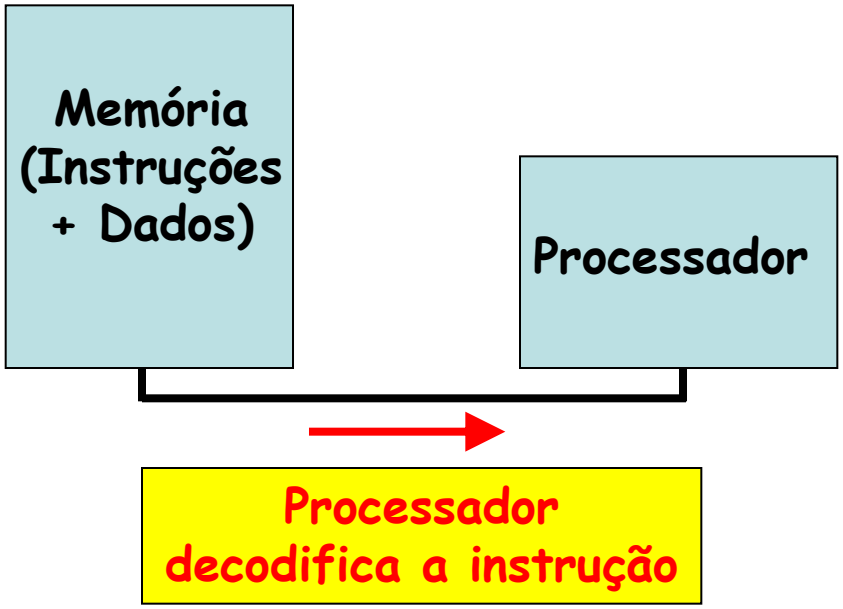
Interpretação

- O **ciclo de execução de um programa** é usado durante o processamento
  - Operação realizada para processar linguagem de nível baixo
    - Ciclo denominado "busca-decodificação-execução"



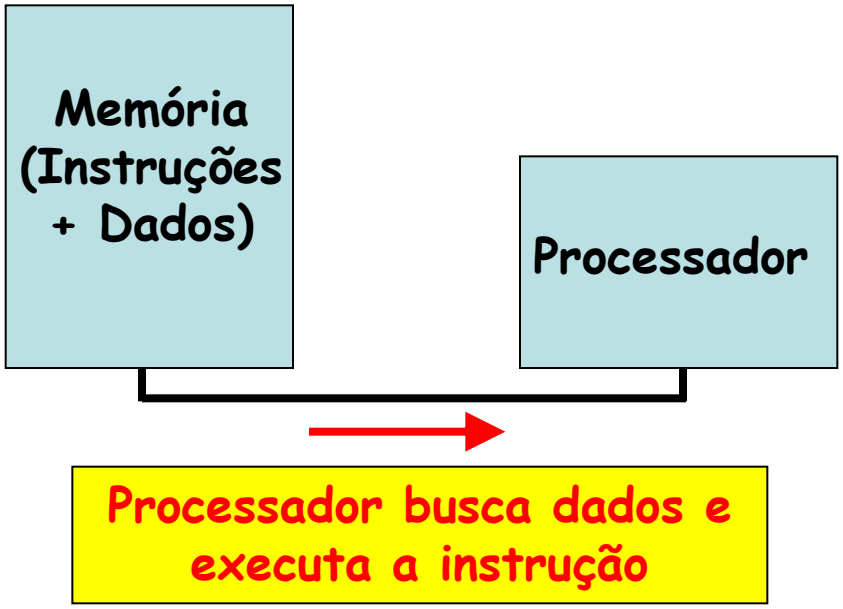
Interpretação

- O **ciclo de execução de um programa** é usado durante o processamento
  - Operação realizada para processar linguagem de nível baixo
    - Ciclo denominado "busca-decodificação-execução"



Interpretação

- O **ciclo de execução de um programa** é usado durante o processamento
  - Operação realizada para processar linguagem de nível baixo
    - Ciclo denominado "busca-decodificação-execução"



Interpretação

- Qual é a relação do ciclo de execução com a interpretação?
  - O ciclo de execução é realizado em um nível mais baixo
    - Com instruções em formato que o processador consegue decodificar
  - A interpretação utiliza o **mesmo processo de "busca-decodificação-execução"** em um nível mais alto
  - Portanto, o processo de interpretação é uma abstração em nível alto de um processo de nível baixo
    - Considera que a linguagem de alto nível é a própria linguagem de baixo nível

Tradução

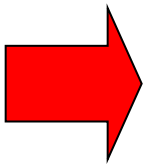
- Programa em linguagem de programação em nível alto é totalmente decodificado em um programa em linguagem de nível baixo
  - Processo realizado **antes** da execução do programa
    - Processo gera um novo programa
    - Programa gerado em nível baixo é equivalente ao programa original em nível alto
- Para isso,
  - Programa é decodificado em um processo chamado de compilação
    - Programa que realiza a compilação é chamado de **compilador**
    - Um compilador para cada arquitetura de processador

Tradução

- A tradução pode ser dividida em duas grandes partes:
  - Análise do programa fonte
    - Dados de entrada
  - Síntese do programa objeto executável

```
#include <stdio.h>
main() {
  int n = 3;
  IF (n > 5) {
    imprimir "n > 5";
  }
  IF (n <= 5) {
          imprimir "n
<= 5";
        }
}
```

Compilação



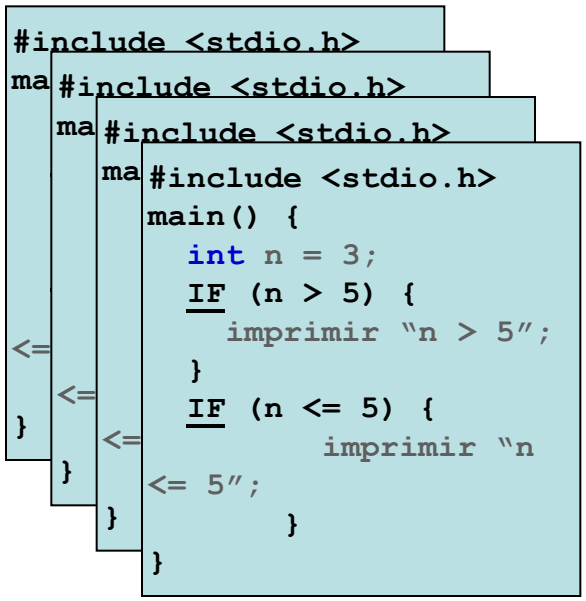
Programa objeto (*.o)

Tradução

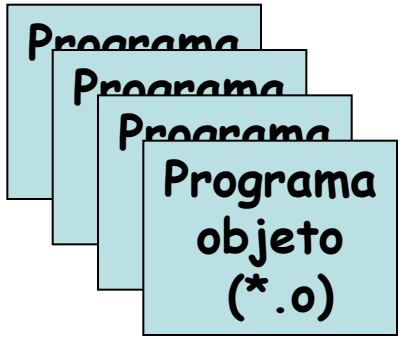
- A saída de um processo de compilação consiste em:
  - Programas objetos (*.o)
    - Programas "quase" executáveis
    - Podem fazer referências a dados externos ou outros programas

- Ligação
  - Realizada por um programa **ligador**
  - Une diversos programas objetos em um único programa executável
    - Um programa em alto nível pode ser composto de diversos subprogramas ou pode fazer referência a programas externos ou ainda pode utilizar funções definidas em bibliotecas externas

Tradução



Compilação



Ligação

Programa executável (*.exe)

Execução

n <= 5

Interpretação X Tradução

- Compiladores e interpretadores dependem da arquitetura do processador (ISA)
  - Intel x86, SPARC, AMD64 etc.
- Os programas interpretados são **sempre reinterpretados** durante a execução
  - **Independente da arquitetura do processador**
    - Em compensação, o desempenho pode ser mais baixo pois todos os comandos são interpretados antes de executar
- Os programas compilados **não precisam ser sempre recompilados**
  - **Torna a execução mais rápida**
    - Porém, dependente da arquitetura do processador

Como Escolher uma Linguagem de Programação

- Envolve sempre dois aspectos
  - Semântica
    - Significado
  - Sintaxe
    - Forma
- Utiliza estruturas básicas de controle
  - Formas naturais de pensar e adequadas à construção de algoritmos inteligíveis

Como Escolher uma Linguagem de Programação?

- Envolve sempre dois aspectos
  - Semântica
    - Significado
  - Sintaxe
    - Forma
- Utiliza estruturas básicas de controle
  - Formas naturais de pensar e adequadas à construção de algoritmos inteligíveis

```
SE condição satisfeita ENTÃO
            execute ação 1
SENÃO
            execute ação 2
```

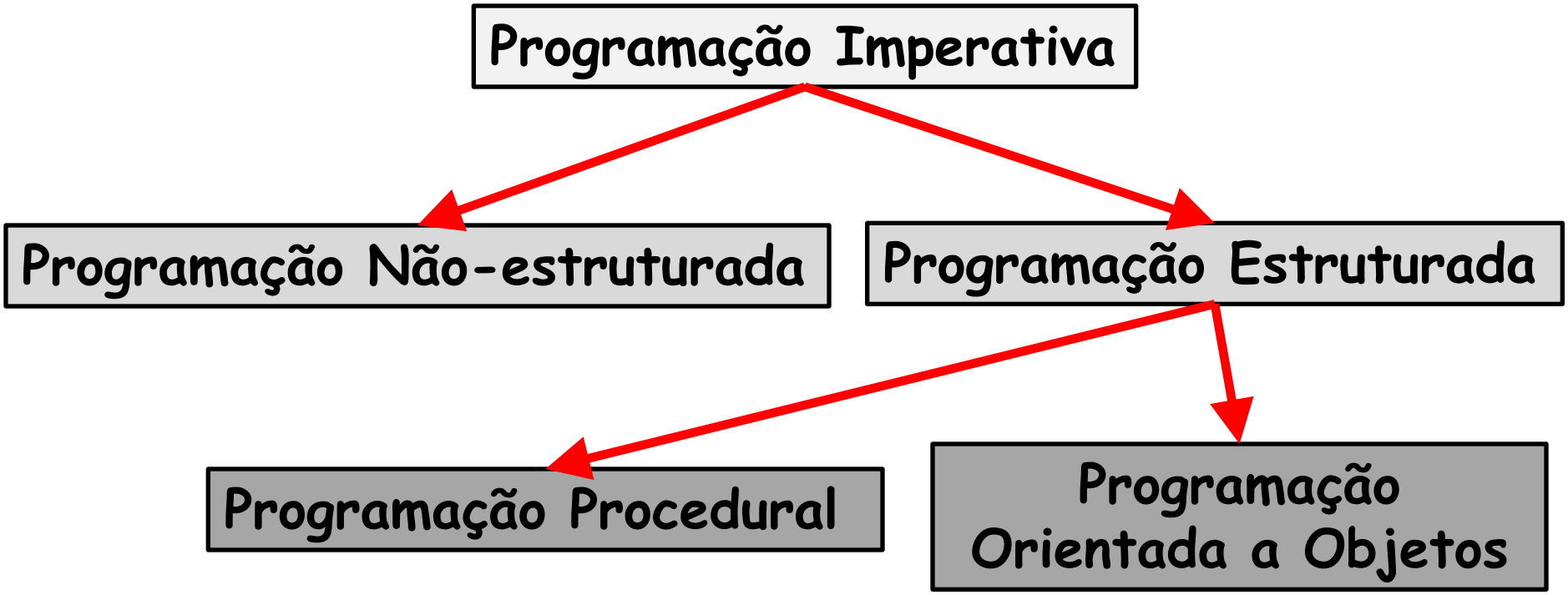
-Semântica: Condição e desvio

-Sintaxe: Maneira como se representa a condição e o desvio

Como Escolher uma Linguagem de Programação?

- Tipo de problema
  - Algumas linguagens são mais adaptadas a determinados tipos de problema do que outras
    - Entretanto, a evolução de uma linguagem a torna mais abrangente
- Fatores fora de controle
  - Super importantes! ☺
    - A linguagem escolhida é a mesma do software a ser utilizado ou a mesma que o programador se sente mais à vontade
      - Ex.: O NS-3 (*Network Simulator* 3) é escrito em C++

Paradigmas de Programação em Alto Nível



Programação Orientada a Objetos

- Programação **imperativa** e **estruturada**, porém...
  - Enquanto a programação procedural é estruturada em...
    - **Procedimentos**
      - Estruturas de dados e algoritmos
  - A programação orientada a objetos é estruturada em...
    - **Classes e objetos**
      - Objetos encapsulam estruturas de dados e procedimentos

Programação Orientada a Objetos

- Programação **imperativa** e **estruturada**, porém...
  - Enquanto a programação procedural é estruturada em...
    - **Procedimentos**
      - Estruturas de dados e algoritmos
  - A programação orientada a objetos é estruturada em...
    - **Classes e objetos**
      - Objetos encapsulam estruturas de dados e procedimentos

Procedural

```
main() {
    define_carro();
    entra_carro();
    sair_carro();
}
```

Orientada a objetos

```
main() {
    Carro carro;
    carro.entrar();
    carro.sair();
}
```

Por que a Linguagem C?

- Permite o desenvolvimento de programas **menores** e mais **rápidos**
  - Programador possui controle maior sobre o código
    - Programador deve:
      - Definir onde armazenar as variáveis na memória
      - Alocar e liberar a memória
      - Trabalhar com endereços de memória
  - Em compensação, a programação é mais detalhada
    - Detalhes que não são "preocupações" em linguagens de mais alto nível como: Linguagens de scripts, Java e C++
- Possui sintaxe simples
  - Não possui muitas funções embutidas

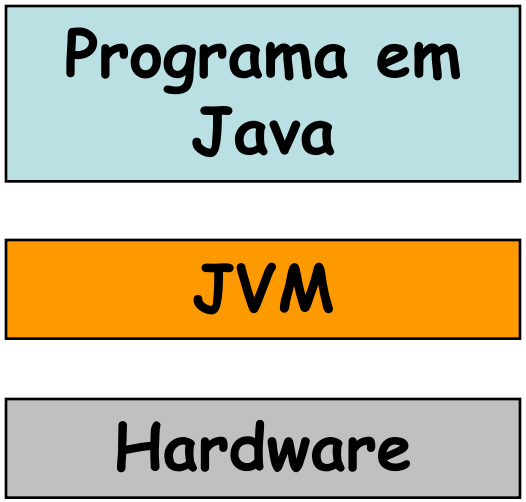
Por que a Linguagem C++?

- Aumento de exigências de mercado
  - **Reuso** de software
    - Modularidade, facilidade de modificação
  - **Rapidez, correção e economia**
    - Programação em mais alto nível
- Programação orientada a objetos
  - Objetos: componentes reutilizáveis de software
    - Modelam itens do mundo real
  - Programas orientados a objetos
    - Mais fáceis de compreender, corrigir e modificar

Estudo de Caso: Linguagem de Programação Java

- Para evitar que todos os programas sejam recompilados para cada arquitetura de processadores
  - O Java utiliza uma máquina virtual (*Java Virtual Machine* - JVM)
  - Assim, os programas são sempre compilados para a mesma arquitetura, a arquitetura da JVM
    - Essa característica torna os programas multiplataformas
      - Funcionam para qualquer arquitetura sem precisar de recompilação
    - Programa compilado chamado de *bytecode*
  - Entretanto, para essa característica ser possível
    - Deve existir uma JVM para cada arquitetura
      - É melhor instalar uma JVM específica e executar qualquer programa do que ter que recompilar cada um dos programas para cada arquitetura

Estudo de Caso: Linguagem de Programação Java



Compilação

- Processo de conversão de um programa em código fonte textual em código de máquina
  - Código em C/C++ ➔ Sequência de 1's e 0's

Programa em C: `hello.c`

Compilação: `gcc -Wall hello.c -o h`

Programa em C++: `hello.cpp`

Compilação: `g++ -Wall hello.cpp -o h`

Compilação

- Opção: -Wall (***Warning all***)
  - Representa uma importante ajuda para detecção de problemas em C/C++
  - Deve ser **sempre** usada
- Formato das mensagens:
  - **arquivo:número_da_linha:mensagem**

```c
#include "stdio.h"

int main () {
        printf ("Dois mais dois é %f\n", 4);
        return 0;
}
```

Erro de execução!!!

```
miguel@pegasus-linux:~$ ./gcc-ex1
Dois mais dois é -1.993767
```

```
miguel@pegasus-linux:~$ gcc -Wall gcc-ex1.c -o gcc-ex1
gcc-ex1.c: In function ‘main’:
gcc-ex1.c:4: warning: format ‘%f’ expects type ‘double’, but argument 2 has type ‘int’
```

Compilação de Múltiplos Arquivos

- Programas podem ser divididos em múltiplos arquivos
  - Simplifica a edição e a compreensão

gcc-ex2.c

```
#include "gcc-hello-ex2.h"

int main () {
        hello ("WORLD");
        return 0;
}
```

gcc-hello-ex2.h

```
void hello (const char *);
```

gcc-hello-ex2.c

```
#include <stdio.h>
#include "gcc-hello-ex2.h"

void hello (const char * nome) {
        printf ("Hello, %s!\n", nome);
}
```

```
miguel@pegasus-linux:~$ gcc -Wall gcc-hello-ex2.c gcc-ex2.c -o gcc-ex2
miguel@pegasus-linux:~$ ./gcc-ex2
Hello, WORLD!
```

Compilação de Múltiplos Arquivos

- Compilação dos arquivos de maneira independente
  - Opção: -c (Compila arquivo em arquivo objeto)
    - Por padrão, nome do arquivo objeto é o mesmo do arquivo fonte, com extensão diferente

```
miguel@pegasus-linux:~$ ls gcc-*ex2.*
gcc-ex2.c  gcc-hello-ex2.c  gcc-hello-ex2.h
miguel@pegasus-linux:~$ gcc -Wall -c gcc-ex2.c
miguel@pegasus-linux:~$ ls gcc-*ex2.*
gcc-ex2.c  gcc-ex2.o  gcc-hello-ex2.c  gcc-hello-ex2.h
miguel@pegasus-linux:~$ gcc -Wall -c gcc-hello-ex2.c
miguel@pegasus-linux:~$ ls gcc-*ex2.*
gcc-ex2.c  gcc-ex2.o  gcc-hello-ex2.c  gcc-hello-ex2.h  gcc-hello-ex2.o
```

Arquivos objetos possuem referências a funções externas, mas não possuem ainda o endereço de memória correspondente que permanece indefinido

Compilação de Múltiplos Arquivos

- Criação de arquivos executáveis
  - Arquivos objetos são ligados (*linked*)
    - GCC usa o programa ligador `ld`
  - Não há necessidade de usar o -Wall
    - Opção já deve ter sido usada para compilar os arquivos fonte e nada mais é alertado

```
miguel@pegasus-linux:~$ gcc gcc-hello-ex2.o gcc-ex2.o -o gcc-ex2
miguel@pegasus-linux:~$ ./gcc-ex2
Hello, WORLD!
```

Processo de ligação preenche os endereços para as funções externas que estavam indefinidos

Compilação de Múltiplos Arquivos

- Recompilação de arquivos fonte
  - Apenas o arquivo fonte alterado precisa ser recompilado e o programa deve ser religado
    - Simplifica a recompilação em casos de programas muito grandes e com muitos arquivos
    - Dispensa a necessidade dos usuários terem todos os arquivos fonte de um mesmo programa

```
#include "gcc-hello-ex2.h"

int main () {
        hello ("TURMA");
        return 0;
}
```

```
miguel@pegasus-linux:~$ gcc -c gcc-ex2.c
miguel@pegasus-linux:~$ gcc gcc-hello-ex2.o gcc-ex2.o -o gcc-ex2
miguel@pegasus-linux:~$ ./gcc-ex2
Hello, TURMA!
```

Makefile

- Especifica um conjunto de regras de compilação
  - Alvos (executáveis) e suas dependências (arquivos objeto e arquivos fonte)

```
alvo: dependências
[TAB] comando
```

- Programa **make** lê a descrição de um projeto no Makefile
  - Para cada alvo, o **make** verifica quando as dependências foram modificadas pela última vez para determinar se o alvo precisa ser reconstruído

Makefile

- Identação da linha de comando é feita com um TAB
- Começa sempre com o primeiro alvo
  - Chamado de objetivo padrão (*default goal*)
- Regras implícitas
  - Regras definidas por padrão
    - Exs.: Arquivos ".o" são obtidos de arquivos ".c" através da compilação

      Arquivos executáveis são obtidos pela ligação de arquivos ".o"
  - Definidas em termos de variáveis do **make**
    - Exs.: CC (compilador C)

      CFLAGS (opções de compilação em programas em C)

Atribuição é do tipo `VARIÁVEL=VALOR`

Makefile

```
CC=gcc
CFLAGS=-Wall

gcc-ex2:
	$(CC) $(CFLAGS) gcc-ex2.c gcc-hello-ex2.c -o gcc-ex2


clean:
	rm -vf gcc-ex2 *.o
```

Se fosse C++, bastaria substituir as variáveis CC para CPP e CFLAGS para CPPFLAGS

```
miguel@pegasus-linux:~$ make
gcc -Wall gcc-ex2.c gcc-hello-ex2.c -o gcc-ex2
miguel@pegasus-linux:~$ ./gcc-ex2
Hello, TURMA!
miguel@pegasus-linux:~$ make clean
rm -vf gcc-ex2 *.o
`gcc-ex2' removido
```

Makefile

```
CC=gcc
CFLAGS=-Wall

gcc-ex2: gcc-ex2.o gcc-hello-ex2.o
	$(CC) gcc-ex2.o gcc-hello-ex2.o -o gcc-ex2

gcc-ex2.o: gcc-ex2.c
	$(CC) $(CFLAGS) -c gcc-ex2.c

gcc-hello-ex2.o: gcc-hello-ex2.c
	$(CC) $(CFLAGS) -c gcc-hello-ex2.c

clean:
	rm -vf gcc-ex2 *.o
```

Com dependências

```
miguel@pegasus-linux:~$ make
gcc -Wall -c gcc-ex2.c
gcc -Wall -c gcc-hello-ex2.c
gcc gcc-ex2.o gcc-hello-ex2.o -o gcc-ex2
miguel@pegasus-linux:~$ ./gcc-ex2
Hello, TURMA!
miguel@pegasus-linux:~$ make clean
rm -vf gcc-ex2 *.o
`gcc-ex2' removido
`gcc-ex2.o' removido
`gcc-hello-ex2.o' removido
```

Makefile

```
CC=gcc
CFLAGS=-Wall

all: gcc-ex2.o gcc-hello-ex2.o
	$(CC) gcc-ex2.o gcc-hello-ex2.o -o gcc-ex2

clean:
	rm -vf gcc-ex2 *.o
```

Com alvo "all"

```
miguel@pegasus-linux:~$ make
gcc -Wall   -c -o gcc-ex2.o gcc-ex2.c
gcc -Wall   -c -o gcc-hello-ex2.o gcc-hello-ex2.c
gcc gcc-ex2.o gcc-hello-ex2.o -o gcc-ex2
miguel@pegasus-linux:~$ ./gcc-ex2
Hello, TURMA!
miguel@pegasus-linux:~$ make clean
rm -vf gcc-ex2 *.o
`gcc-ex2' removido
`gcc-ex2.o' removido
`gcc-hello-ex2.o' removido
```

Makefile

```
CC=gcc
CFLAGS=-Wall

OBJECTS = gcc-ex2.o gcc-hello-ex2.o

all: $(OBJECTS)
	$(CC) $(OBJECTS) -o gcc-ex2

gcc-ex2.o: gcc-ex2.c
	$(CC) $(CFLAGS) -c gcc-ex2.c

gcc-hello-ex2.o: gcc-hello-ex2.c
	$(CC) $(CFLAGS) -c gcc-hello-ex2.c

clean:
	rm -vf gcc-ex2 *.o
```

Com macros

```
miguel@pegasus-linux:~$ make
gcc -Wall -c gcc-ex2.c
gcc -Wall -c gcc-hello-ex2.c
gcc gcc-ex2.o gcc-hello-ex2.o -o gcc-ex2
miguel@pegasus-linux:~$ ./gcc-ex2
Hello, TURMA!
miguel@pegasus-linux:~$ make clean
rm -vf gcc-ex2 *.o
`gcc-ex2' removido
`gcc-ex2.o' removido
`gcc-hello-ex2.o' removido
```

Makefile

- Macros especiais
  - $@
    - Nome completo do alvo atual
  - $?
    - Lista de arquivos desatualizados para dependência atual
  - $<
    - Arquivo fonte da única dependência atual

Makefile

Com macros especiais

```
CC=gcc
CFLAGS=-Wall

OBJECTS = gcc-ex2.o gcc-hello-ex2.o

PROGRAM = gcc-ex2

all: $(PROGRAM)

$(PROGRAM): $(OBJECTS)
	$(CC) $(OBJECTS) -o $@

.c.o:
	$(CC) $(CFLAGS) -c $<

clean:
	rm -vf gcc-ex2 *.o
```

```
miguel@pegasus-linux:~$ make
gcc -Wall -c gcc-ex2.c
gcc -Wall -c gcc-hello-ex2.c
gcc gcc-ex2.o gcc-hello-ex2.o -o gcc-ex2
miguel@pegasus-linux:~$ ./gcc-ex2
Hello, TURMA!
miguel@pegasus-linux:~$ make clean
rm -vf gcc-ex2 *.o
`gcc-ex2' removido
`gcc-ex2.o' removido
`gcc-hello-ex2.o' removido
```

Bibliotecas Externas

- Conjuntos de arquivos objetos pré-compilados
  - Podem ser ligados aos programas
- Provêem funções ao sistema
  - Ex.: Função `sqrt` da biblioteca matemática do C
- Armazenadas em arquivos especiais com extensão ".a"
  - Referenciados como bibliotecas estáticas
  - Criados de arquivos objetos com uma ferramenta chamada GNU archiver, o `ar`
  - Encontrados em `/usr/lib` e `/lib`
    - Ex.: Biblioteca matemática: `/usr/lib/libm.a`

      Biblioteca padrão C: /usr/lib/libc.a, contém funções como o printf e é ligada a todos os programas por padrão

Bibliotecas Externas

- Arquivo **math.h**
  - Contém os protótipos das funções da biblioteca matemática em /usr/lib/libm.a
    - No diretório padrão **/usr/include/**

```c
#include <math.h>
#include <stdio.h>

int main () {
        double x = sqrt (2.0);
        printf ("A raiz quadrada de 2.0 eh %f\n", x);
        return 0;
}
```

-l*NOME* é um atalho para /usr/lib/lib*NOME*.a

```
miguel@pegasus-linux:~$ gcc -Wall gcc-ex3.c /usr/lib/libm.a -o gcc-ex3
miguel@pegasus-linux:~$ ./gcc-ex3
A raiz quadrada de 2.0 eh 1.414214
miguel@pegasus-linux:~$ gcc -Wall gcc-ex3.c -lm -o gcc-ex3
miguel@pegasus-linux:~$ ./gcc-ex3
A raiz quadrada de 2.0 eh 1.414214
```

Bibliotecas Externas

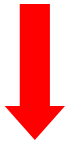
- Definição das funções deve aparecer primeiro que as funções propriamente ditas
  - Logo, as bibliotecas devem aparecer depois dos arquivos fontes ou arquivos objetos

```
gcc -Wall -lm gcc-ex3.c -o gcc-ex3
```

```
gcc -Wall gcc-ex3.c -lm -o gcc-ex3
```

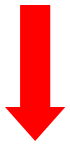
Opções de Compilação

- Diretórios padrão de arquivos de cabeçalho
  - `/usr/local/include`
  - `/usr/include`



Ordem de busca (precedência)

- Diretórios padrão de bibliotecas
  - `/usr/local/lib`
  - `/usr/lib`



Ordem de busca (precedência)

Se um arquivo de cabeçalho for encontrado em `/usr/local/include`, o compilador não busca mais em `/usr/include`. O mesmo acontece para as bibliotecas.

Opções de Compilação

- Opção –I:
  - Adiciona novo diretório para o início do caminho de busca por arquivos de cabeçalho
- Opção –L:
  - Adiciona novo diretório para o início do caminho de busca por bibliotecas

Assumindo que o programa utilize um sistema de banco de dados (GNU Database Management - GDBM):

```
gcc -Wall -I/opt/gdbm-1.8.3/include -L/opt/gdbm-1.8.3/lib
gcc-ex4.c -lgdbm
```

Variáveis de Ambiente

- Ao invés de sempre adicionar os diretórios, pode-se definir variáveis de ambiente
  - Variáveis podem ser definidas em arquivos do shell

Arquivos de cabeçalho:

```
C_INCLUDE_PATH=/opt/gdbm-1.8.3/include
export C_INCLUDE_PATH
```

Se fosse C++...

```
CPLUS_INCLUDE_PATH=/opt/gdbm-1.8.3/include
export CPLUS_INCLUDE_PATH
```

Bibliotecas:

```
LIBRARY_PATH=/opt/gdbm-1.8.3/lib
export LIBRARY_PATH
```

Assim...

```
gcc -Wall gcc-ex4.c -lgbdm -o gcc-ex4
```

Variáveis de Ambiente

- Para inserir múltiplos diretórios...

```
gcc -Wall -I. -I/opt/gdbm-1.3.8/include -I/net/include -L. -
L/opt/gdbm-1.3.8/lib -L/net/lib gcc-ex4.c -lgdbm -lnet -o
gcc-ex4
```

- Através das variáveis de ambiente...

```
C_INCLUDE_PATH=.:/opt/gdbm-1.3.8/include:/net/include
LIBRARY_PATH=.:/opt/gdbm-1.3.8/lib:/net/lib

gcc -Wall gcc-ex4.c -lgbdm -lnet -o gcc-ex4
```

Bibliotecas Estáticas e Compartilhadas

- Bibliotecas estáticas (arquivos ".a")
  - Copia da biblioteca o código de máquina das funções externas usadas pelo programa
    - Copia no executável final durante a ligação

Bibliotecas Estáticas e Compartilhadas

- Bibliotecas compartilhadas (arquivos ".so")
  - Arquivo executável final contém apenas uma tabela das funções que necessita, ao invés de todo código de máquina
    - Menor arquivo executável final
  - Antes do arquivo executável começar a rodar, o código de máquina das funções externas é copiado para a **memória** pelo Sistema Operacional
    - Transferido do arquivo da biblioteca compartilhada que está em disco
      - Processo chamado de ligação dinâmica (*dynamic linking*)
  - No Windows as bibliotecas *.so são as *.dll

Bibliotecas Estáticas e Compartilhadas

- Bibliotecas compartilhadas (arquivos ".so")
  - Uma vez carregadas em memória...
    - Podem ser compartilhadas por todos os programas em execução que utilizam a mesma biblioteca
  - Por padrão, o gcc busca primeiro a biblioteca compartilhada `lib`***`NOME`***`.so` para depois buscar a `lib`***`NOME`***`.a`
    - Ocorre sempre que o a biblioteca `-l`***`NOME`*** é adicionada pelo gcc
  - Variável de ambiente para bibliotecas compartilhadas são carregadas na `LD_LIBRARY_PATH`:
    - `LD_LIBRARY_PATH=/opt/gdbm-1.3.8/lib`

Bibliotecas Estáticas e Compartilhadas

- Opção –static:
  - Força o uso das bibliotecas estáticas
    - Pode ser vantajoso para evitar a definição de **LD_LIBRARY_PATH**

```
gcc -Wall -static -I/opt/gdbm-1.3.8/include -L/opt/gdbm-1.3.8/lib gcc-ex4.c -lgdbm -o gcc-ex4
```

- Outra maneira de forçar o uso de uma determinada biblioteca é colocando o caminho completo...

```
gcc -Wall -I/opt/gdbm-1.3.8/include gcc-ex4.c /opt/gdbm-1.3.8/lib/libgdbm.a -o gcc-ex4
```

Compilação em Modo de Depuração

- Arquivo executável
  - Não contém referências ao código fonte do programa original
    - Exs.: nomes de variáveis, número de linhas etc.
  - É simplesmente uma sequência de instruções em código de máquina criado pelo compilador

Como então seria possível fazer depuração se não há como encontrar, de maneira simples, possíveis fontes de erro no código fonte?

Compilação em Modo de Depuração

- Opção: -g
  - Opção para depuração
    - Armazena informação adicional de depuração em arquivos executáveis e em arquivos objetos
    - Depuradores conseguem rastrear erros baseados em informações contidas nos arquivos compilados

Compilação em Modo de Depuração

- Opção: *-g*
  - Opção para depuração
    - Informações de nomes e linhas de código são armazenadas em tabelas de símbolos contidas nos arquivos compilados
    - Quando programa pára de maneira anormal, o compilador gera um arquivo chamado `core` que combinado com informações do modo de depuração auxiliam na busca da causa do erro
      - Informações da linha que provocou o erro e do valor das variáveis

Compilação em Modo de Depuração

```
int foo (int *p);

int main () {

    int *p = 0;
    return foo (p);
}

int foo (int *p) {


    int y = *p;
    return y;
}
```

Qual é o erro no código ao lado?

Compilação em Modo de Depuração

```c
int foo (int *p);

int main () {
    // Ponteiro inicializado com 0
    int *p = 0;
    return foo (p);
}

int foo (int *p) {
    // Função tenta desreferenciar um
    // ponteiro nulo, o que gera erro
    int y = *p;
    return y;
}
```

```
miguel@pegasus-linux:~$ gcc -Wall -g coreDump.c -o coreDump
miguel@pegasus-linux:~$ ./coreDump
Falha de segmentação
miguel@pegasus-linux:~$ ulimit -c
0
miguel@pegasus-linux:~$ ulimit -c unlimited
miguel@pegasus-linux:~$ ulimit -c
unlimited
miguel@pegasus-linux:~$ ./coreDump
Falha de segmentação (core dumped)
```

Compilação em Modo de Depuração

- **`ulimit -c`**
  - Exibe o tamanho permitido para arquivos do tipo core
- **`ulimit -c unlimited`**
  - Permite tamanho ilimitado para arquivos do tipo core

Depurador

- Programa gdb (GNU Debugger)
  - **gdb <arquivo_executável> <arquivo_core>**

```
miguel@pegasus-linux:~$ gdb coreDump core
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...

warning: Can't read pathname for load map: Input/output error.
Reading symbols from /lib/i686/cmov/libc.so.6...done.
Loaded symbols for /lib/i686/cmov/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
Core was generated by `./coreDump'.
Program terminated with signal 11, Segmentation fault.
[New process 3204]
#0  0x080483a9 in foo (p=0x0) at coreDump.c:12
12              int y = *p;
(gdb) print y
$1 = 134518124
(gdb) print p
$2 = (int *) 0x0
```

Comandos gdb

- **`print <variável>`**
  - Imprime o valor da variável no momento da parada do programa
- **`trace`**
  - Imprime as chamadas de funções e os valores das variáveis até a parada do programa
    - Procedimento chamado de backtrace

Comandos gdb

```
This GDB was configured as "i486-linux-gnu"...

warning: Can't read pathname for load map: Input/output error.
Reading symbols from /lib/i686/cmov/libc.so.6...done.
Loaded symbols for /lib/i686/cmov/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
Core was generated by `./coreDump'.
Program terminated with signal 11, Segmentation fault.
[New process 3204]
#0  0x080483a9 in foo (p=0x0) at coreDump.c:12
12              int y = *p;
(gdb) print y
$1 = 134518124
(gdb) print p
$2 = (int *) 0x0
(gdb) break main
Breakpoint 1 at 0x8048385: file coreDump.c, line 5.
(gdb) run
Starting program: /home/miguel/coreDump

Breakpoint 1, main () at coreDump.c:5
5               int *p = 0;
```

Comandos gdb

- Inserção de breakpoint
  - Execução do programa pára e retorna o controle para o depurador
    - **`break <função, linha ou posição de memória>`**
- Execução do programa
  - Programa começa a execução até encontrar o breakpoint
    - **`run`**
    - Para continuar a execução usar o **`step`** ou **`next`**
      - Comando step mostra a execução de qualquer função chamada na linha (maior nível de detalhes)

Comandos gdb

```
(gdb) print p
$2 = (int *) 0x0
(gdb) break main
Breakpoint 1 at 0x8048385: file coreDump.c, line 5.
(gdb) run
Starting program: /home/miguel/coreDump

Breakpoint 1, main () at coreDump.c:5
5               int *p = 0;
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/miguel/coreDump

Breakpoint 1, main () at coreDump.c:5
5               int *p = 0;
(gdb) next
6               return foo (p);
(gdb) next

Program received signal SIGSEGV, Segmentation fault.
0x080483a9 in foo (p=0x0) at coreDump.c:12
12              int y = *p;
```

Comandos gdb

- Valores de variáveis podem ser corrigidas
  - `set variable <variável> = <valor>`
    - Após isso o programa executará normalmente mesmo após utilizar o comando step

- **`finish`**
  - Continua a execução da função até o final, exibindo o valor de retorno encontrado

- **`continue`**
  - Continua a execução do programa até o final ou até o próximo breakpoint

Comandos gdb

- Programas com loop infinito podem ser também depurados
  - Primeiro executa o programa
  - Depois inicia o **gdb** para o programa em execução
    - **`attach <pid>`**: Anexa um determinado programa em execução ao gdb
      - Comando "**`ps x`**" obtém identificador do processo
    - **`kill`**: Mata o programa em execução

Comandos gdb

```c
int main () {
        unsigned int i = 0;

        while (1) i++;

        return 0;
}
```

```
miguel@pegasus-linux:~$ gcc -Wall -g gcc-ex9.c -o gcc-ex9
miguel@pegasus-linux:~$ ./gcc-ex9
Morto
```

```
miguel@pegasus-linux:/mnt/ufrj/disciplinas/linguagens/projetos$ gdb gcc-ex9
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...
(gdb) attach 3850
Attaching to program: /mnt/ufrj/disciplinas/linguagens/projetos/gcc-ex9, process
 3850
Reading symbols from /lib/i686/cmov/libc.so.6...done.
Loaded symbols for /lib/i686/cmov/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
0x08048390 in main () at gcc-ex9.c:4
4               while (1) i++;
(gdb) print i
$1 = 1349516083
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb)
```

Comandos gdb

```c
int main () {
        unsigned int i = 0;

        while (1) i++;

        return 0;
}
```

```
miguel@pegasus-linux:~$ gcc -Wall -g gcc-ex9.c -o gcc-ex9
miguel@pegasus-linux:~$ ./gcc-ex9
Morto
```

```
miguel@pegasus-linux:/mnt/ufrj/disciplinas/linguagens/projetos$ gdb gcc-ex9
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...
(gdb) attach 3850
Attaching to program: /mnt/ufrj/disciplinas/linguagens/projetos/gcc-ex9, process
 3850
Reading symbols from /lib/i686/cmov/libc.so.6...done.
Loaded symbols for /lib/i686/cmov/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
0x08048390 in main () at gcc-ex9.c:4
4               while (1) i++;
(gdb) print i
$1 = 1349516083
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb)
```

pid do processo

Otimizações

- O GCC é um compilador para fazer otimizações
  - Possui opções para aumentar a velocidade e reduzir o tamanho dos executáveis gerados
- Processo de otimização é complexo
  - Código em alto nível pode ser traduzido em diferentes combinações de instruções para atingir o mesmo resultado
  - Código gerado depende do processador
    - Ex.: há processadores que oferecem um número maior de registradores que outros

Otimização no Código

- Não requer conhecimento da arquitetura do processador
  - Dois tipos comuns:
    - Eliminação de expressão comum (*common subexpression elimination*)
    - Inserção de funções (*function inlining*)

Eliminação de Expressão Comum

- Consiste em calcular uma expressão no código fonte com menos instruções
  - Reusa resultados já calculados

```
x = cos(v)*(1 + sen(u/2)) + sen(w)*(1 - sen(u/2))

Pode ser reescrito, da seguinte maneira...

t = sen(u/2)
x = cos(v)*(1 + t) + sen(w)*(1 - t)
```

Inserção de Funções

- Sempre que uma função é chamada...
  - CPU armazena os argumentos da função em registradores e/ou posições de memória apropriados
  - Há um pulo para o início da função trazendo as páginas na memória virtual para a memória física ou cache
  - Início da execução do código da função
  - Retorno ao ponto original de execução após o término da função

Todo esse procedimento é chamado de sobrecarga de chamada de função que consome tempo de processamento!

Inserção de Funções

- Elimina a sobrecarga de chamada de função
  - Substitui a chamada pelo próprio código da função
  - É mais significativa caso a função chamada seja pequena
    - Código inserido no programa possui um número menor de instruções que o necessário para realizar a chamada

Exemplo de código cuja sobrecarga da chamada seria comparável ao tempo de execução necessário para realizar uma única multiplicação

```
double sq (double x) {
       return x*x;
}
```

Inserção de Funções

- Elimina a sobrecarga de chamada de função
  - Substitui a chamada pelo próprio código da função
  - É mais significativa caso a função chamada seja pequena
    - Código inserido no programa possui um número menor de instruções que o necessário para realizar a chamada

E se a função fosse usada dessa maneira???

```
for (i = 0; i < 10000000; i++) {
       sum += sq (i + 0.5);
}
```

Inserção de Funções

- Elimina a sobrecarga de chamada de função
  - Substitui a chamada pelo próprio código da função
  - É mais significativa caso a função chamada seja pequena
    - Código inserido no programa possui um número menor de instruções que o necessário para realizar a chamada

O procedimento de inserção (*inlining*) da função resulta no seguinte código otimizado:

```
for (i = 0; i < 10000000; i++) {
        double t = (i + 0.5); //Variável temporária
        sum += t*t;
}
```

Inserção de Funções

- O GCC seleciona funções para serem inseridas
  - Baseado no tamanho das funções
- Funções também podem ser solicitadas para serem inseridas explicitamente pelo programador
  - Uso da palavra-chave `inline`
    - Entretanto, compilador verifica se há a possibilidade

Compromisso entre Velocidade e Tamanho

- Algumas opções de otimização podem tornar o código:
  - Mais rápido, mas...
  - Maior em tamanho

Equivalentemente

  - Menor em tamanho, mas...
  - Mais lento

Ex.: desenrolamento de laços (*loop unrolling*)

Desenrolamento de Laços

- Aumenta a velocidade de execução do programa
  - Elimina a verificação da condição de término do laço em cada iteração

```
for (i = 0; i < 8; i++) {
        y[i] = i;
}
```

Otimização

```
y[0] = 0;
y[1] = 1;
y[2] = 2;
y[3] = 3;
y[4] = 4;
y[5] = 5;
y[6] = 6;
y[7] = 7;
```

Processo de otimização elimina a necessidade de checagem da condição de contorno e ainda permite paralelização das sentenças de atribuição. Entretanto, aumenta o tamanho do arquivo se o número de atribuições for maior que o tamanho do laço.

Agendamento

- Nível mais baixo de otimização
  - Compilador determina a melhor ordem das instruções individuais
    - Ordem deve ser tal que todas as instruções possuam os dados necessários antes da execução
- Não aumenta o tamanho do executável
  - Mas demora mais tempo para compilar devido a maior complexidade do procedimento

Níveis de Otimização

- Oferecidos pelo GCC para lidar com:
  - Tempo de compilação
  - Uso de memória
  - Compromisso entre velocidade e tamanho do executável

Níveis de Otimização

- Escolhidos com uma opção de linha de comando
  - Formato: **-O*NÍVEL*** (Nível pode variar de 0 até 3)
    - -O0: *GCC* não realiza otimizações
    - -O1: *GCC* realiza as formas mais comuns de otimizações que não requerem compromisso entre velocidade e tamanho
    - -O2: *GCC* adiciona otimizações em relação ao 01 que incluem agendamento de instruções
    - -O3: *GCC* adiciona otimizações em relação ao 02 que incluem inserção de funções e procedimentos que aumentam a velocidade mas aumentam o tamanho do código. **Por esse motivo, normalmente o O2 se torna a melhor opção**

Níveis de Otimização

- Escolhidos com uma opção de linha de comando
  - Formato: `-O`*`NÍVEL`* (Nível pode variar de 0 até 3)
    - funroll-loops: Independente das anteriores, habilita o desenrolamento de laços
    - `-Os`: *GCC* seleciona apenas otimizações que reduzem o tamanho do executável para plataformas com restrições de recursos

Níveis de Otimização

```c
#include <stdio.h>

double powern (double d, unsigned n) {
        double x = 1.0;
        unsigned j;

        for (j = 1; j <= n; j++)
                x *= d;

        return x;
}

int main () {
        double sum = 0.0;
        unsigned i;

        for (i = 1; i <= 100000000; i++) {
                sum += powern (i, i % 5);
        }

        printf ("sum = %g\n", sum);

        return 0;
}
```

Níveis de Otimização

```
miguel@pegasus-linux:~$ gcc -Wall -O0 gcc-ex8.c -o gcc-ex8 -lm
miguel@pegasus-linux:~$ time ./gcc-ex8
sum = 4e+38

real    0m9.179s
user    0m8.977s
sys     0m0.040s
miguel@pegasus-linux:~$ gcc -Wall -O1 gcc-ex8.c -o gcc-ex8 -lm
miguel@pegasus-linux:~$ time ./gcc-ex8
sum = 4e+38

real    0m6.977s
user    0m6.744s
sys     0m0.060s
miguel@pegasus-linux:~$ gcc -Wall -O2 gcc-ex8.c -o gcc-ex8 -lm
miguel@pegasus-linux:~$ time ./gcc-ex8
sum = 4e+38

real    0m5.855s
user    0m5.688s
sys     0m0.044s
```

`user:` **tempo de execução do processo em CPU**

`total:` tempo total para a execução do programa incluindo tempo de espera por CPU

`sys:` tempo esperando chamadas de sistema

Níveis de Otimização

```
miguel@pegasus-linux:~$ gcc -Wall -O3 gcc-ex8.c -o gcc-ex8 -lm
miguel@pegasus-linux:~$ time ./gcc-ex8
sum = 4e+38

real    0m5.840s
user    0m5.664s
sys     0m0.040s
miguel@pegasus-linux:~$ gcc -Wall -O3 -funroll-loops gcc-ex8.c -o gcc-ex8 -lm
miguel@pegasus-linux:~$ time ./gcc-ex8
sum = 4e+38

real    0m4.851s
user    0m4.700s
sys     0m0.024s
```

Os diferentes níveis de otimização permitiram uma queda de tempo de execução de quase 50% (-O0: 8,977s para –O3 – funroll-loops: 4,700s)
Em compensação, o tamanho do arquivo executável passou de 6,5k para 6,8k

Compilação de Programas em C++

- GCC compila códigos fonte em C++ diretamente em código de máquina
  - Outros compiladores convertem o código primeiro para C para depois compilar
- Procedimento é o mesmo para compilação em C
  - Uso do comando g++ ao invés do gcc
    - É parte do *GNU Compiler Collection*, assim como o gcc
    - Inclui bibliotecas típicas de C++
      - Ex.: iostream ao invés de stdio.h para E/S de dados
    - Possui mensagens de warning específicas
      - -Wall avisa sobre métodos e funções virtuais

Ferramentas Relacionadas ao Compilador

- Criação de bibliotecas **estáticas**: **`ar`** (*GNU archiver*)
  - Combina uma coleção de arquivos objeto em um único arquivo de biblioteca (arquivo *archive*)
    - Uma biblioteca é uma maneira conveniente de distribuir um número grande de arquivos objetos relacionados em um único arquivo

```
ar <nome_da_biblioteca> <lista_de_arquivos_objeto>
```

Ferramentas Relacionadas ao Compilador

Arquivo: gcc-hello-ex2.h

```
void hello (const char *);
void bye ();
```

Arquivo: gcc-bye-ex2.c

```
#include <stdio.h>
#include "gcc-hello-ex2.h"

void bye () {
        printf ("Good Bye!\n");
}
```

Arquivo: gcc-hello-ex2.c

```
#include <stdio.h>
#include "gcc-hello-ex2.h"

void hello (const char * nome) {
        printf ("Hello, %s!\n", nome);
}
```

Arquivo: gcc-ex2.c

```
#include "gcc-hello-ex2.h"

int main () {
        hello ("TURMA");
        bye ();
        return 0;
}
```

Ferramentas Relacionadas ao Compilador

```
miguel@pegasus-linux:~$ gcc -Wall -c gcc-hello-ex2.c -o gcc-hello-ex2.o
miguel@pegasus-linux:~$ gcc -Wall -c gcc-bye-ex2.c -o gcc-bye-ex2.o
miguel@pegasus-linux:~$ ar cr libhello.a gcc-hello-ex2.o gcc-bye-ex2.o
miguel@pegasus-linux:~$ ar t libhello.a
gcc-hello-ex2.o
gcc-bye-ex2.o
miguel@pegasus-linux:~$ gcc -Wall gcc-ex2.c libhello.a -o gcc-ex2
miguel@pegasus-linux:~$ ./gcc-ex2
Hello, TURMA!
Good Bye!
miguel@pegasus-linux:~$ gcc -Wall -L. gcc-ex2.c -lhello -o gcc-ex2
miguel@pegasus-linux:~$ ./gcc-ex2
Hello, TURMA!
Good Bye!
```

Opções do `ar`:
`cr`: create and replace
`t`: table of contents

Ferramentas Relacionadas ao Compilador

- Dependências de bibliotecas **compartilhadas**: `ldd` (*LD Dependencies*)
  - Verifica quais são e se as bibliotecas compartilhadas necessárias já foram encontradas
    - Caso tenham sido, o caminho da biblioteca é apresentado

```
ldd <arquivo_executável>
```

Ferramentas Relacionadas ao Compilador

- Como criar as bibliotecas compartilhadas (dinâmicas)
  - Opção **-shared**

```
gcc -shared -o <biblioteca.so> <lista_arquivos_objetos>
```

  - Outra alternativa:
    - Opção **-Wl**: lista opções ao programa ligador separadas por vírgulas

```
gcc -shared -Wl,-soname,<biblioteca.so.versão> -o
<biblioteca.so.versão_completa> <lista_arquivos_objetos>
```

Opções do ligador (`ld`):
`-soname`: quando um executável é ligado a uma biblioteca compartilhada, o ligador dinâmico tenta ligar o executável com a biblioteca de nome passado com a opção `soname` e não com a biblioteca com o nome dado com o `-o`. Isso permite a existência de bibliotecas com nomes e versões diferentes da criada.

Ferramentas Relacionadas ao Compilador

Arquivo: gcc-hello-ex2.h

```
void hello (const char *);
void bye ();
```

Arquivo: gcc-bye-ex2.c

```
#include <stdio.h>
#include "gcc-hello-ex2.h"

void bye () {
        printf ("Good Bye!\n");
}
```

Arquivo: gcc-hello-ex2.c

```
#include <stdio.h>
#include "gcc-hello-ex2.h"

void hello (const char * nome) {
        printf ("Hello, %s!\n", nome);
}
```

Arquivo: gcc-ex2.c

```
#include "gcc-hello-ex2.h"

int main () {
        hello ("TURMA");
        bye ();
        return 0;
}
```

Ferramentas Relacionadas ao Compilador

```
miguel@pegasus-linux:~$ gcc -Wall -c gcc-hello-ex2.c -o gcc-hello-ex2.o
miguel@pegasus-linux:~$ gcc -Wall -c gcc-bye-ex2.c -o gcc-bye-ex2.o
miguel@pegasus-linux:~$ gcc -shared -o libhello.so gcc-hello-ex2.o gcc-bye-ex2.o
miguel@pegasus-linux:~$ gcc -Wall -L. gcc-ex2.c -o gcc-ex2 -lhello
miguel@pegasus-linux:~$ ./gcc-ex2
./gcc-ex2: error while loading shared libraries: libhello.so: cannot open shared object file: No suc
h file or directory
miguel@pegasus-linux:~$ ldd gcc-ex2
        linux-gate.so.1 =>  (0xb7fd8000)
        libhello.so => not found
        libc.so.6 => /lib/i686/cmov/libc.so.6 (0xb7e6a000)
        /lib/ld-linux.so.2 (0xb7fd9000)
miguel@pegasus-linux:~$ echo $LD_LIBRARY_PATH
:/home/miguel/simulacao/ns-allinone-2.34/otcl-1.13:/home/miguel/simulacao/ns-allinone-2.34/lib
miguel@pegasus-linux:~$ export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
miguel@pegasus-linux:~$ echo $LD_LIBRARY_PATH
.::/home/miguel/simulacao/ns-allinone-2.34/otcl-1.13:/home/miguel/simulacao/ns-allinone-2.34/lib
miguel@pegasus-linux:~$ ldd gcc-ex2
        linux-gate.so.1 =>  (0xb7f06000)
        libhello.so => ./libhello.so (0xb7f02000)
        libc.so.6 => /lib/i686/cmov/libc.so.6 (0xb7d96000)
        /lib/ld-linux.so.2 (0xb7f07000)
miguel@pegasus-linux:~$ ./gcc-ex2
Hello, TURMA!
Good Bye!
```

Ferramentas Relacionadas ao Compilador

- Diferenças entra o uso de bibliotecas dinâmicas e estáticas...

```
miguel@pegasus-linux:~$ gcc -Wall -static -L. gcc-ex2.c -o gcc-ex2 -lhello
miguel@pegasus-linux:~$ ll -h gcc-ex2
-rwxr-xr-x 1 miguel miguel 553K Mar 24 10:38 gcc-ex2
miguel@pegasus-linux:~$
miguel@pegasus-linux:~$ gcc -Wall -L. gcc-ex2.c -o gcc-ex2 -lhello
miguel@pegasus-linux:~$ ll -h gcc-ex2
-rwxr-xr-x 1 miguel miguel 6,6K Mar 24 10:38 gcc-ex2
```

Tamanho do executável é bem maior ao utilizar bibliotecas estáticas!

Ferramentas Relacionadas ao Compilador

- Medida de desempenho: **valgrind**
  - Conjunto de programas para depurar e avaliar o desempenho do programa
    - A ferramenta mais utilizada é a **Memcheck** para avaliar erros comuns de memória

```
gcc -Wall -g <arquivo.c> -o <arquivo.o>

Programa deve ser compilado em modo de depuração

valgrind --leak-check=yes <arquivo_executável>

A ferramenta Memcheck é usada por padrão. A opção leak-check
liga o detector de vazamento de memória
```

Ferramentas Relacionadas ao Compilador

```c
#include <stdlib.h>

void f(void) {
        int *x = malloc(10 * sizeof(int));
        x[10] = 0;
}

int main(void) {
        f();
        return 0;
}
```

Quais são os problemas desse programa?

Ferramentas Relacionadas ao Compilador

```c
#include <stdlib.h>

void f(void) {
        int *x = malloc(10 * sizeof(int));
        x[10] = 0;      // problema 1: Alocação de memória errada
}                       // problema 2: Vazamento de memória, x não foi liberado

int main(void) {
        f();
        return 0;
}
```

Quais são os problemas desse programa?

Ferramentas Relacionadas ao Compilador

```c
#include <stdlib.h>

void f(void) {
        int *x = malloc(10 * sizeof(int));
        x[10] = 0;      // problema 1: Alocação de memória errada
}                       // problema 2: Vazamento de memória, x não foi liberado

int main(void) {
        f();
        return 0;
}
```

```
miguel@pegasus-linux:~$ gcc -Wall -g gcc-ex13.c -o gcc-ex13
miguel@pegasus-linux:~$ valgrind --leak-check=yes ./gcc-ex13
==4450== Memcheck, a memory error detector.
==4450== Copyright (C) 2002-2007, and GNU GPL'd, by Julian Seward et al.
==4450== Using LibVEX rev 1854, a library for dynamic binary translation.
==4450== Copyright (C) 2004-2007, and GNU GPL'd, by OpenWorks LLP.
==4450== Using valgrind-3.3.1-Debian, a dynamic binary instrumentation framework.
==4450== Copyright (C) 2000-2007, and GNU GPL'd, by Julian Seward et al.
==4450== For more details, rerun with: -v
==4450==
==4450== Invalid write of size 4
==4450==    at 0x80483BF: f (gcc-ex13.c:5)
==4450==    by 0x80483DC: main (gcc-ex13.c:9)
==4450==  Address 0x4194050 is 0 bytes after a block of size 40 alloc'd
==4450==    at 0x4023D6E: malloc (vg_replace_malloc.c:207)
==4450==    by 0x80483B5: f (gcc-ex13.c:4)
==4450==    by 0x80483DC: main (gcc-ex13.c:9)
==4450==
==4450== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 13 from 1)
==4450== malloc/free: in use at exit: 40 bytes in 1 blocks.
==4450== malloc/free: 1 allocs, 0 frees, 40 bytes allocated.
==4450== For counts of detected errors, rerun with: -v
==4450== searching for pointers to 1 not-freed blocks.
==4450== checked 60,204 bytes.
```

Ferramentas Relacionadas ao Compilador

```c
#include <stdlib.h>

void f(void) {
        int *x = malloc(10 * sizeof(int));
        x[10] = 0;      // problema 1: Alocação de memória errada
}                       // problema 2: Vazamento de memória, x não foi liberado

int main(void) {
        f();
        return 0;
}
```

Erro de alocação detectado!

```
miguel@pegasus-linux:~$ gcc -Wall -g gcc-ex13.c -o gcc-ex13
miguel@pegasus-linux:~$ valgrind --leak-check=yes ./gcc-ex13
==4450== Memcheck, a memory error detector.
==4450== Copyright (C) 2002-2007, and GNU GPL'd, by Julian Seward et al.
==4450== Using LibVEX rev 1854, a library for dynamic binary translation.
==4450== Copyright (C) 2004-2007, and GNU GPL'd, by OpenWorks LLP.
==4450== Using valgrind-3.3.1-Debian, a dynamic binary instrumentation framework.
==4450== Copyright (C) 2000-2007, and GNU GPL'd, by Julian Seward et al.
==4450== For more details, rerun with: -v
==4450==
==4450== Invalid write of size 4
==4450==    at 0x80483BF: f (gcc-ex13.c:5)
==4450==    by 0x80483DC: main (gcc-ex13.c:9)
==4450==  Address 0x4194050 is 0 bytes after a block of size 40 alloc'd
==4450==    at 0x4023D6E: malloc (vg_replace_malloc.c:207)
==4450==    by 0x80483B5: f (gcc-ex13.c:4)
==4450==    by 0x80483DC: main (gcc-ex13.c:9)
==4450==
==4450== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 13 from 1)
==4450== malloc/free: in use at exit: 40 bytes in 1 blocks.
==4450== malloc/free: 1 allocs, 0 frees, 40 bytes allocated.
==4450== For counts of detected errors, rerun with: -v
==4450== searching for pointers to 1 not-freed blocks.
==4450== checked 60,204 bytes.
```

Ferramentas Relacionadas ao Compilador

```c
#include <stdlib.h>

void f(void) {
        int *x = malloc(10 * sizeof(int));
        x[10] = 0;      // problema 1: Alocação de memória errada
}                       // problema 2: Vazamento de memória, x não foi liberado

int main(void) {
        f();
        return 0;
}
```

```
==4450==
==4450==
==4450== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==4450==    at 0x4023D6E: malloc (vg_replace_malloc.c:207)
==4450==    by 0x80483B5: f (gcc-ex13.c:4)
==4450==    by 0x80483DC: main (gcc-ex13.c:9)
==4450==
==4450== LEAK SUMMARY:
==4450==    definitely lost: 40 bytes in 1 blocks.
==4450==      possibly lost: 0 bytes in 0 blocks.
==4450==    still reachable: 0 bytes in 0 blocks.
==4450==         suppressed: 0 bytes in 0 blocks.
```

Ferramentas Relacionadas ao Compilador

```c
#include <stdlib.h>

void f(void) {
        int *x = malloc(10 * sizeof(int));
        x[10] = 0;      // problema 1: Alocação de memória errada
}                       // problema 2: Vazamento de memória, x não foi liberado

int main(void) {
        f();
        return 0;
}
```

```
==4450==
==4450==
==4450== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==4450==    at 0x4023D6E: malloc (vg_replace_malloc.c:207)
==4450==    by 0x80483B5: f (gcc-ex13.c:4)
==4450==    by 0x80483DC: main (gcc-ex13.c:9)
==4450==
==4450== LEAK SUMMARY:
==4450==    definitely lost: 40 bytes in 1 blocks.
==4450==      possibly lost: 0 bytes in 0 blocks.
==4450==    still reachable: 0 bytes in 0 blocks.
==4450==         suppressed: 0 bytes in 0 blocks.
```

Erro de vazamento detectado!
O "definitely lost" significa que o programa está realmente vazando memória, se ele tivesse dúvida, colocaria "probably lost"

Processo de Compilação

- Processo com múltiplos estágios
  - Envolve diferentes ferramentas
    - Próprio compilador (`gcc` ou `g++`)
    - Montador (`as`)
    - Ligador (`ld`)
  - O processo completo ao se invocar o compilador é:
    - Pré-processamento ➔ compilação ➔ montagem ➔ ligação
      - Processo completo pode ser visto com a opção `-v`

Processo de Compilação

Pré-processamento:
cpp <arquivo.c> > <arquivo.i>

```c
#include <stdio.h>

int main () {
        printf ("Hello World!\n");
        return 0;
}
```

```c
extern int feof (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern int ferror (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern void clearerr_unlocked (FILE *__stream) __attribute__ ((__nothrow__));
extern int feof_unlocked (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern int ferror_unlocked (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern void perror (__const char *__s);
# 1 "/usr/include/bits/sys_errlist.h" 1 3 4
# 27 "/usr/include/bits/sys_errlist.h" 3 4
extern int sys_nerr;
extern __const char *__const sys_errlist[];
# 823 "/usr/include/stdio.h" 2 3 4
extern int fileno (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern int fileno_unlocked (FILE *__stream) __attribute__ ((__nothrow__)) ;
# 842 "/usr/include/stdio.h" 3 4
extern FILE *popen (__const char *__command, __const char *__modes) ;
extern int pclose (FILE *__stream);
extern char *ctermid (char *__s) __attribute__ ((__nothrow__));
# 882 "/usr/include/stdio.h" 3 4
extern void flockfile (FILE *__stream) __attribute__ ((__nothrow__));
extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__));
# 912 "/usr/include/stdio.h" 3 4

# 2 "gcc-ex11.c" 2

int main () {
 printf ("Hello World!\n");
 return 0;
}
```

Processo de Compilação

Compilação:
`gcc -Wall -S <arquivo.i>`

```
extern int feof (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern int ferror (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern void clearerr_unlocked (FILE *__stream) __attribute__ ((__nothrow__));
extern int feof_unlocked (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern int ferror_unlocked (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern void perror (__const char *__s);
# 1 "/usr/include/bits/sys_errlist.h" 1 3 4
# 27 "/usr/include/bits/sys_errlist.h" 3 4
extern int sys_nerr;
extern __const char *__const sys_errlist[];
# 823 "/usr/include/stdio.h" 2 3 4
extern int fileno (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern int fileno_unlocked (FILE *__stream) __attribute__ ((__nothrow__)) ;
# 842 "/usr/include/stdio.h" 3 4
extern FILE *popen (__const char *__command, __const char *__modes) ;
extern int pclose (FILE *__stream);
extern char *ctermid (char *__s) __attribute__ ((__nothrow__));
# 882 "/usr/include/stdio.h" 3 4
extern void flockfile (FILE *__stream) __attribute__ ((__nothrow__));
extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__)) ;
extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__));
# 912 "/usr/include/stdio.h" 3 4

# 2 "gcc-ex11.c" 2

int main () {
 printf ("Hello World!\n");
 return 0;
}
```

```
        .file   "gcc-ex11.c"
        .section        .rodata
.LC0:
        .string "Hello World!"
        .text
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $4, %esp
        movl    $.LC0, (%esp)
        call    puts
        movl    $0, %eax
        addl    $4, %esp
        popl    %ecx
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (Debian 4.3.2-1.1) 4.3.2"
        .section        .note.GNU-stack,"",@progbits
```

Processo de Compilação

```
        .file   "gcc-ex11.c"
        .section        .rodata
.LC0:
        .string "Hello World!"
        .text
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $4, %esp
        movl    $.LC0, (%esp)
        call    puts
        movl    $0, %eax
        addl    $4, %esp
        popl    %ecx
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (Debian 4.3.2-1.1) 4.3.2"
        .section        .note.GNU-stack,"",@progbits
```

Montagem:
`as <arquivo.s> -o <arquivo.o>`

Arquivo .o em linguagem de máquina

Processo de Compilação

Ligação:
`gcc <arquivo.o>`

Arquivo .o → **Executável**

```
miguel@pegasus-linux:~$ cpp gcc-ex11.c > gcc-ex11.i
miguel@pegasus-linux:~$ gcc -Wall -S gcc-ex11.i
miguel@pegasus-linux:~$ as gcc-ex11.s -o gcc-ex11.o
miguel@pegasus-linux:~$ gcc gcc-ex11.o -o gcc-ex11
miguel@pegasus-linux:~$ ./gcc-ex11
Hello World!
```

Leitura Recomendada

- Capítulo 1 dos livros
  - Deitel, "*C++ How to Program*", 5th edition, Editora Prentice Hall, 2005
  - Waldemar Celes, Renato Cerqueira e José Lucas Rangel, "Introdução a Estrutura de Dados com Técnica de Programação em C", Editora Campus-Elsevier, 2004
- Brian Gough, "*An Introduction to GCC*", 2nd edition, **Network Theory Ltd**, 2005
- "GNU `make'", disponível em **http://www.gnu.org/software/make/manual/make.html**
- "Static, Shared Dynamic and Loadable Linux Libraries", disponível em **http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html**