Utilitário GMAKE
Gerador de diretivas MAKE
versão 6.0

Arndt von Staa

LES - Laboratório de Engenharia de Software
Departamento de Informática, PUC-Rio
Fevereiro 2006

1. Objetivo

É objetivo do `GMAKE` gerar os arquivos a seguir, a partir de diretivas de *composição do construto* e de diretivas de *definição da plataforma de desenvolvimento*:

- Arquivos contendo *scripts* para o utilitário `MAKE`,
- Arquivos contendo *scripts* para o ligador `LINK`
- Arquivos contendo *scripts* para gerenciador de bibliotecas `LIB`
- Arquivos contendo relatórios de estatísticas e de referências cruzadas relativo ao produto

Os arquivos de *script* orientam o `MAKE`, o ligador `LINK`, ou o gerente de bibliotecas, a construir ou reconstruir o construto[1], utilizando para isto uma plataforma de desenvolvimento específica. O *script* de `MAKE` é gerado explorando todos os arquivos de código fonte que compõem o produto a construir. Finalmente, o *script* de `MAKE` pode envolver uma variedade de tipos de arquivos possivelmente processados ou gerados por ferramentas disponibilizadas pelo usuário.

2. Motivação

Criar um arquivo *script* para `MAKE`, identificando exatamente todas as dependências, é um trabalho enfadonho e sujeito a erros, em particular ao se praticar desenvolvimento incremental. Por outro lado, ao desenvolver sistemas com um número grande de módulos, é interessante utilizar diretivas exatas, uma vez que muitas vezes isto minimiza a necessidade de recompilação. Um *script* exato pode ser gerado, percorrendo-se, recursivamente, todos os arquivos que compõem um determinado artefato[2] objetivo, e respectivos arquivos de inclusão (e.g. `#include`) utilizados, identificando-se, assim, todos os arquivos dependentes utilizados para gerar esse artefato.

---

[1] ***Construto*** é um programa que implementa parcial ou totalmente um determinado conjunto de requisitos. Um construto destina-se à avaliação dessa implementação parcial, permitindo a antecipação da identificação de possíveis problemas de especificação ou implementação. No limite um construto corresponde ao ***produto*** almejado. Construtos correspondem a ***incrementos*** quando se utiliza ***desenvolvimento incremental***.

[2] ***Artefato*** é qualquer resultado tangível do desenvolvimento. São exemplos, módulos, programas, bibliotecas, documentos, e sistemas compostos por vários programas. Um artefato a ser distribuído para o cliente é um ***produto***.

Artefatos podem ser gerados a partir de arquivos de código fonte e de tabelas, utilizando uma variedade de compiladores, transformadores e/ou ferramentas desenvolvidas pelo usuário. Além disso, os artefatos que compõem um determinado construto podem estar escritos em várias linguagens de representação. Conseqüentemente, podem participar da construção ou reconstrução do artefato, tanto compiladores e ligadores, geradores de documentação, como uma variedade de ferramentas desenvolvidas pela equipe de desenvolvimento do construto objetivo. Além disso, os arquivos a serem processados podem estar distribuídos sobre uma estrutura de diretórios complexa. Finalmente, diferentes plataformas de desenvolvimento põem à disposição programas `MAKE`, processadores de linguagem, ligadores e gerentes de bibliotecas, cada qual com uma interface própria, obrigando a criação de arquivos *script* especializados para cada plataforma de desenvolvimento utilizada.

O programa `GMAKE` é paramétrico e permite a geração de *scripts* para `MAKE` para uma variedade de plataformas de desenvolvimento e de naturezas de produto. O programa cria um arquivo `.MAKE` contendo o *script* para o utilitário `MAKE`, e, opcionalmente, um arquivo `.BUILD` contendo um *script* para o ligador `LINK` ou para o gerador de bibliotecas de módulos `LIB`.

A sua configuração inicial prevê a geração de diretivas para o `MAKE` que acompanha os compiladores MS-Visual C/C++ versões 6 em diante. Modificando as diretivas de geração da plataforma, podem-se gerar diretivas `MAKE` para outras plataformas de desenvolvimento. Através da edição dos arquivos de diretivas, pode-se alterar tanto a plataforma de desenvolvimento, como a natureza, as linguagens e as ferramentas utilizadas para produzir os componentes do artefato objetivo.

2.1. Requisitos da versão 6.0

- **sistema operacional**: A versão compilada roda em Windows 2000, XP ou mais recente. O programa `GMAKE` é um programa comando de linha e deve ser ativado em uma janela `CMD`, de outra forma perdem-se os resultados exibidos.
- **memória principal**: o número de módulos que constituem um construto é limitado pelo tamanho da memória disponível. Caso ocorra erro por falta de memória, particione o projeto do produto em dois ou mais subprojetos, *n - 1* dos quais geram bibliotecas, e o *n-ésimo* gera o construto objetivo.

2.2. Restrições da versão 6

- A estrutura de diretórios deve estar toda contida em um único disco.
- Os nomes de arquivos e diretórios (pastas) não podem envolver caracteres em branco.
- Os nomes de arquivos e diretórios serão todos transformados em letras minúsculas. Caso os nomes de arquivos e diretórios sejam sensíveis à caixa na plataforma usada, todos deverão ser escritos em minúsculas. Em Windows não existe a restrição.

2.3. Exemplos de cenários de uso

- Gerar *scripts* para compilar e ligar módulos de um programa redigidos em C e/ou C++, sendo que os módulos podem estar distribuídos sobre uma estrutura de diretórios complexa, devendo ser compilados e ligados pelo compilador XX. Deve ser possível gerar *scripts* capazes de, sob controle de algum parâmetro de linha de comando passado para o `MAKE`, compilar alguns módulos para teste detalhado,

compilar todo o construto para teste não detalhado ou compilar de forma otimizada o conjunto para produção.

- Gerar *scripts* para compilar e ligar módulos de um programa redigido em uma variedade de linguagens de programação e ferramentas, e que utilizam uma variedade de tabelas geradas, sendo que os módulos fonte e as tabelas fonte podem estar distribuídos sobre uma estrutura de diretórios. O *script* gerado pode utiliza ferramentas desenvolvidas pelo usuário.
- Gerar *scripts* para compilar e criar bibliotecas estáticas ou dinâmicas de módulos objeto.
- Gerar *scripts* para a geração de documentação técnica em formato HTML a partir de um conjunto de módulos C++ contendo comentários marcados.

3. Estrutura padrão de diretórios

O programa GMAKE assume que os módulos, arquivos de inclusão, de definição ou de dados possam estar distribuídos sobre um ou mais diretórios. O *default* é que tudo esteja em um mesmo diretório. Recomenda-se que estes diretórios formem uma estrutura padronizada.

Um exemplo de estrutura padronizada de diretórios:

..\<prod>	contém os diversos componentes de um produto complexo de nome <prod>.
..\<prod>\<subprod$_1$>	contém os artefatos que constituem o subproduto 1 de nome <subprod$_1$>.
..\<prod>\<subprod$_1$>\composicao	contém os *scripts* de composição .comp dos diversos construtos, os relatórios de composição .list e os *scripts* do MAKE .make.
..\<prod>\<subprod$_1$>\cpp	contém os módulos C++ de implementação .cpp, e de definição .hpp.
..\<prod>\<subprod$_1$>\c	contém os módulos C de implementação .c, e de definição .h.
..\<prod>\<subprod$_1$>\tabelas	contém as tabelas de definição .incxxx e as tabelas de dados .tabxxx que definem constantes de interface a serem incluídas em módulos a serem compilados, sendo que xxx indica o tipo de tabela.
..\<prod>\<subprod$_1$>\def	contém tabelas fonte, por exemplo tabelas de especificação de *strings* .defstr a partir das quais são geradas outras tabelas (.incstr e .tabstr) a serem incluídas ao compilar módulos. A geração será realizada por ferramentas desenvolvidas pelo usuário.
..\<prod>\<subprod$_1$>\obj	contém os módulos objeto .obj compilados, os *scripts* de ligação .build, o arquivo de recursos .res compilado e bibliotecas estáticas .lib.

`..\<prod>\<subprod1>\produto` contém o programa final `.exe` composto e todos os demais arquivos necessários para o sua correta execução uso, como, por exemplo bibliotecas dinâmicas `.dll` e arquivos de parâmetros para o construto `.parm`.

`..\<prod>\<subprod1>\teste` contém os *scripts* de teste automatizado `.script`, os *logs* dos testes `.log` e estatísticas dos testes `.estat`.

`..\<prod>\<subprod1>\doc` contém os artefatos de documentação.

Cada produto deve estar em um diretório específico. Idealmente cada componente desse produto deve estar em um subdiretório. São exemplos de componentes: bibliotecas (`.LIB` ou `.DLL`) e os respectivos módulos e tabelas que as constituem. Organizando-se os arquivos de um sistema desta forma, reduz-se o número de arquivos por diretório, cria-se um particionamento lógico do conjunto de arquivos, simplificando a busca e a manipulação do conjunto de arquivos que formam o sistema. Permite também o desenvolvimento de ferramentas que eliminem o lixo que freqüentemente se acumula nos diretórios.

Faz parte da definição de composição de um produto, a definição da estrutura de diretórios padrão. Recomenda-se que todos os diretórios e *paths* sejam definidos de forma relativa. Desta forma torna-se fácil armazenar cópias dos arquivos de um sistema em várias máquinas, ou centralizá-las em um servidor de rede.

4. Utilização

4.1. Ativação de GMAKE

Para executar utilize o comando:

```
GMAKE /C<ArqConst> [/P<ArqPlat>] [/B<DirBase>] [/L<ArqList>]
          { /? | /H }
```

<`DirBase`**>** *opcional*, é o nome do diretório base do projeto do construto a produzir. Caso não seja fornecido, `DirBase` será o diretório corrente ao ativar `GMAKE`. O diretório `DirBase` deve ser o diretório corrente ao ativar o `MAKE` para produzir o construto em questão.

`<ArqConst>` *obrigatório*, é o nome do arquivo de diretivas para a geração dos arquivos *script* para `MAKE`, `LINK` ou `LIB`, e que descreve o ***construto*** a ser composto aplicando `MAKE` ao arquivo de diretivas gerado. `<ArqConst>` será procurado relativo ao diretório base. O nome de extensão *default* é `.comp`.

`<ArqPlat>` *opcional*, é o nome do arquivo de diretivas que define a ***plataforma*** de desenvolvimento em uso. Para cada plataforma a ser utilizada deverá ser criado um arquivo de diretivas `<ArqPlat>`. Na prática é conveniente criar um arquivo de diretivas da plataforma para cada projeto. Este arquivo definirá a plataforma do projeto de todos os construtos. De maneira geral, um mesmo construto descrito pelo arquivo `<ArqConst>` poderá ser criado em diversas plataformas bastando determinar qual o arquivo de diretivas de plataforma a utilizar. Caso este parâmetro de linha de comando não seja fornecido,

será procurado o arquivo de diretivas *default* GMAKE.DAT no diretório base. O nome de extensão *default* é .parm.

<ArqList> *opcional*, é o nome do arquivo que conterá o ***relatório de geração***. Este relatório fornece a lista de referências cruzadas de dependência de todos os arquivos e as estatísticas de dimensão dos arquivos. O nome de extensão *default* é .list. Veja a seção 4.2.1 Itens da seção [Diretorios] para mais detalhes quanto à forma de definir este arquivo.

Os parâmetros de linha de comando podem ser fornecidos em qualquer ordem. O programa GMAKE valida os parâmetros, lê as diretivas contidas nos arquivos e valida estas diretivas. Os parâmetros /C, /P, /L e /B podem ser letras maiúsculas ou minúsculas. As mensagens de erro serão exibidas no vídeo. Ocorrendo erro, a geração dos arquivos *script* é inibida.

Para obter um auxílio de uso resumido, utilize o comando:

```
GMAKE /? ou GMAKE ? ou GMAKE /h ou GMAKE /H
```

Após a validação dos dados, é apresentado, no vídeo, um resumo da geração a ser realizada.

Após executar, o programa GMAKE retorna:

0 - se executou corretamente.

2 - se encontrou solicitação de auxílio.

4 - se encontrou algum erro de dados ou de execução. As mensagens de erro estarão sinalizadas por linhas começando com os caracteres "***". As de advertência estarão sinalizadas por linhas começando com os caracteres "---".

4.2. Composição do arquivo descritor do construto <ArqConst>

O arquivo <ArqConst> contém a descrição do construto a ser gerado ao aplicar MAKE ao arquivo de diretivas .make gerado. Conforme definido nas diretivas do construto, o *script* .make pode aplicar LINK ou LIB ao arquivo .build gerado.

O arquivo de diretivas é composto por *linhas de comentário* e *linhas de diretivas*. Uma linha de comentário é identificada por um caractere '#' na *primeira* posição da linha. Linhas de comentário e linhas em branco podem ocorrer em qualquer lugar e sempre serão ignoradas.

As linhas de diretivas são agregadas em seções[3]. Cada seção tem um nome de seção. O nome de uma seção tem o formato "[<Nome de seção>]" e deve ser redigido a partir da primeira posição da linha. As demais linhas correspondem às diretivas da seção. As seções podem aparecer em qualquer ordem, exceto as seções [Diretorios] que deverá ser sempre a primeira, e [Fim] que sempre será a última seção do arquivo. Qualquer coisa após [Fim] será tratado como comentário. Os nomes das seções devem ser redigidos tal como aparecem neste documento. GMAKE é sensível à caixa das letras, e as letras não devem ser acentuadas.

[3] Optamos por uma linguagem *ad hoc* similar aos *scripts* de inicialização freqüentemente encontrados, por ser uma linguagem clara, simples de implementar e utilizar, e portátil. Como os *scripts* não necessitam servir de comunicação entre diferentes ferramentas, também não há necessidade do uso de uma linguagem independente de aplicação tal como XML.

Descreveremos a seguir as seções do arquivo <ArqConst>. O capítulo 5 apresenta o esquema do arquivo *script* gerado, salientando onde serão incluídas as diretivas de cada seção. O capítulo 6 apresenta o esquema do arquivo *script* para LINK e LIB.

[Diretorios] Seção obrigatória e necessariamente a primeira seção do arquivo <ArqConst>. Define o nome do construto e a estrutura padrão de diretórios. A estrutura de diretórios é definida, associando *paths* aos nomes de extensão dos arquivos utilizados para gerar o construto. Um *path* define um ou mais diretórios, separados por ponto e vírgula (";"). Mais adiante, na seção 4.2.1, página 9, será detalhado o conteúdo desta seção.

```
[Diretorios]
Nome         =  MeuPrograma
PathDefault  =  ..\Fontes
c            =  ..\Fontes
h            =  ..\Fontes
obj          =  ..\Objetos
def          =  ..\Tabelas
tab          =  ..\Tabelas
inc          =  ..\Tabelas;..\Chaves
```

Figura 1. Exemplo de uma seção Diretorios

[Modulos] Seção obrigatória. Contém a lista de arquivos origem (em geral arquivos contendo módulos de implementação), a partir dos quais deve ser gerado o construto, bem como indicações se o correspondente módulo objeto deve ou não participar de LINK ou LIB. Mais adiante, na seção 4.2.2 página 11, será detalhado o conteúdo desta seção.

```
[Modulos]
str_tst.defstr  g
str_vm.defstr   g
BCDArit         /D_DEBUG
Counters
```

Figura 2. Fragmento de uma seção Modulos

[MacrosAntes] Seção opcional. As linhas de diretivas da seção serão inseridas no arquivo *script* .make, antes das macros padronizadas da plataforma a ser usada. As macros da plataforma são extraídas do arquivo <ArqPlat> e contém as diretivas da plataforma aplicáveis a todos os construtos. As linhas desta seção de diretivas são inseridas em linha nova imediatamente após a última linha gerada até o momento. Esta seção é utilizada, tipicamente, para definir macros *default* ou para atribuir valores a variáveis de controle. Caso o utilitário MAKE a ser utilizado não permita definir nomes na linha de comando, pode-se optar por utilizar esta seção para definir os nomes especiais.

[MacrosApos] Seção opcional. As linhas de diretivas desta seção serão inseridas no arquivo *script* .make logo após às macros padronizadas da plataforma. As linhas de diretivas são inseridas em linha nova imediatamente após a última linha gerada até o momento e precedidas de uma linha em branco. Esta seção é utilizada, tipicamente, para definir macros complementares ou de substituição a outras previamente definidas.

Por exemplo, se for desejado estender o *path* de inclusão a ser utilizado pelo compilador e definir uma variável para o diretório `..\teste` que deverá receber as mensagens de erro:

```
[MacrosApos]
INCLUDE     = $(INCLUDE);..\Tabelas
DERR        = ..\Teste
```

Figura 3. Fragmento de uma seção `Macros`

`[GeraMais]` Seção opcional. As linhas de diretivas da seção serão inseridas no texto de geração dos módulos objetivo, após o último módulo objetivo do produto e antes da definição de geração final do construto (usualmente a ativação do `LINK`), identificada pelo nome "`Construto:`". A inserção se dá ao final da última linha de módulos objetivo, permitindo a inclusão de um marcador de fim de linha "`\`". Esta seção é utilizada, tipicamente, para acrescentar arquivos objetivo a serem construídos pelo `MAKE`, e que, por alguma razão não podem figurar na lista de módulos a processar. Usualmente requer as seções `[ArtefatosAntes]` ou `[ArtefatosApos]` descritas mais adiante. Módulos incluídos na seção `[GeraMais]` não fazem parte do *script* `.build` a menos que sejam incluídos nas seções `[BuildInicio]` ou `[BuildFim]`.

`[LimpaMais]` Seção opcional. As linhas de diretivas da seção serão inseridas após a diretiva "`limpa`" do arquivo `.make` e antes do primeiro módulo a ser gerado. Esta seção é utilizada, tipicamente, para acrescentar ações incondicionais a serem realizadas antes de iniciar a construção do primeiro módulo objetivo. A seção de limpeza é efetuada antes do primeiro módulo ser processado pelo `MAKE`.

`[ArtefatosAntes]` Seção opcional. As linhas de diretivas desta seção são inseridas após a diretiva de limpeza do construto. A inserção se dá em linha nova imediatamente após uma linha em branco inserida após a última linha gerada até o momento. Esta seção é utilizada, tipicamente, para completar a construção de arquivos objeto. Para surtir efeito, os módulos destino (*target*) que aparecem nesta seção devem ter sido declarados na seção `[GeraMais]`.

`[ArtefatosApos]` Seção opcional. As linhas de diretivas desta seção são inseridas após às diretivas de construção do último artefato do construto. A inserção se dá em linha nova imediatamente após uma linha em branco inserida após à última linha gerada até o momento. Esta seção é utilizada, tipicamente, para completar a construção de arquivos objeto. Para surtir efeito, os módulos destino (*target*) que aparecem nesta seção devem ter sido declarados na seção `[GeraMais]`.

`[SemLink]` Seção opcional, mutuamente exclusiva com a seção `[Link]` a seguir. Se aparecer, não será gerado o arquivo *script* `.build`. A seção `[SemLink]` pode ser utilizada para finalizar uma construção caso esta não gere um programa ou biblioteca. Caso se deseje meramente recompilar os módulos fonte, a seção `[SemLink]` pode ser fornecida vazia.

`[Link]` Seção opcional, mutuamente exclusiva com a seção `[SemLink]` anterior. As diretivas desta seção são inseridas, formando os comandos de composição final do construto. Estas diretivas sucedem arquivo objetivo identificado pelo nome "`Construto:`" contido no *script*

`.make`. Esta seção é utilizada para gerar diretivas para ativar o compositor do construto objetivo final. Em geral este compositor será o ligador ou o programa de manutenção de bibliotecas. Caso esta seção não seja fornecida e também não exista a seção `[SemLink]`, serão utilizadas as diretivas `[DefaultLink]` definidas no arquivo `<ArqPlat>`. Pelo fato de se usar o nome `Construto:` como elemento final, o construto será sempre religado, mesmo se nenhum dos objetos tenha sido recompilado.

```
[LINK]
    cd $(Fobj)
    LIB $(L) @$(NOME).build >> $(Ferr)\$(NOME).err
```

Figura 4. Fragmento de uma seção `Build` para gerar uma biblioteca estática

`[BuildInicio]` Seção opcional. As diretivas desta seção contém as linhas do *script* que devem anteceder a lista de módulos componentes do artefato a ser composto pelo `LINK` ou `LIB`. Estas diretivas serão gravadas no arquivo `<construto>.build`. Estas diretivas informam tipicamente os parâmetros de composição a serem utilizados pelo `LINK` ou `LIB`.

```
[BuildInicio]
/OUT:..\Produto\Teste.exe
/INCREMENTAL:NO
/MACHINE:IX86
```

Figura 5. Fragmento de uma seção `BuildInicio` para gerar um programa executável

```
[BuildInicio]
/OUT:..\Produto\Teste.dll
/INCREMENTAL:NO
/MACHINE:IX86
/DLL
/DEF:Teste.def
```

Figura 6. Fragmento de uma seção `BuildInicio` para gerar uma biblioteca dinâmica

```
[BuildInicio]
/OUT:TesteLib.lib
/MACHINE:IX86
```

Figura 7. Fragmento de uma seção `BuildInicio` para gerar uma biblioteca estática

`[BuildFim]` Seção opcional. As diretivas desta seção contém as linhas que sucedem a lista de módulos componentes do artefato a ser composto pelo `LINK` ou `LIB`. Estas diretivas serão gravadas no arquivo `<construto>.build`. Elas informam tipicamente as bibliotecas a serem utilizadas ao compor o artefato.

```
[BuildFim]
TesteLib.lib
```

Figura 8. Fragmento de uma seção `BuildFim` para incluir uma biblioteca estática

`[Fim]` Seção obrigatória e necessariamente a última. Indica término de comandos do arquivo de diretivas. Todas as linhas após `[Fim]` serão tratadas como comentário.

4.2.1. Itens da seção [Diretorios]

Esta seção é necessariamente a primeira seção do arquivo de diretivas do produto. O seu objetivo é definir o nome do construto e a estrutura de diretórios utilizada pelo projeto do construto. Para tornar o *script* .make portátil entre estações de desenvolvimento, recomenda-se que a estrutura de subdiretórios do produto sendo desenvolvido seja padronizada. Desta forma o *script* poderá ser transportado facilmente de uma estação de desenvolvimento para outra, bastando que todos os desenvolvedores utilizem a mesma estrutura de subdiretórios para o produto sendo desenvolvido. Cabe salientar que a padronização da estrutura de subdiretórios de um produto traz consigo uma série de vantagens adicionais, uma vez que permite a criação de outras ferramentas e macros (*batches*) que necessitem explorar os arquivos que formam o construto.

A estrutura de diretórios é definida em termos de *paths*. Um *path* é uma lista de um ou mais diretórios separados por ponto e vírgula. Utilize de preferência nomes relativos para os nomes de diretórios. Os nomes dos diretórios devem ser sempre relativos ao diretório que contém o *script* .make. Os diretórios relativos serão sempre calculados a partir do diretório base do construto DirBase ou, caso este não tenha sido definido, a partir do diretório corrente ao ativar GMAKE (ver seção *Ativação de GMAKE*). Ao dar partida no MAKE para interpretar um determinado *script* .make, assegure-se que o diretório corrente seja o diretório base utilizado ao gerar o *script*.

Cada linha de diretivas da seção [Diretorios] tem o formato:

```
<NomeItem> = <ValorItem>
```

Os itens podem vir em qualquer ordem. Linhas em branco serão ignoradas.

Nome obrigatório, define o nome dos arquivos *script* a serem gerados.

- o arquivo *script* para MAKE será <Nome>.make
- o arquivo *script* para LINK e para LIB será <Nome>.build
- o arquivo de listagem será <Nome>.list, caso não tenha sido definido nem o parâmetro de linha de comando /L nem o atributo ArquivoListagem, ver a seguir.

PathDefault opcional, define o *path* onde devem ser procurados os arquivos cujo nome de extensão não tenha sido associado a algum *path* específico. Caso não seja definido será utilizado o diretório base do construto. Este item é sensível à caixa, deve ser redigido exatamente como apresentado aqui.

xxx opcional, define o *path* associado ao nome de extensão .xxx. ***Os nomes de extensão (exceto o*** PathDefault***) devem ser sempre redigidos usando letras minúsculas.*** Todos os arquivos com nome de extensão igual a .xxx serão procurados em um dos diretórios deste *path*. O primeiro dos diretórios contido no *path* será o diretório onde serão armazenados os arquivos gerados pelos compiladores ou pelas ferramentas do usuário. Os controles de *path* (Pxxx) e diretório (*folder* - Fxxx, identificando o primeiro nome de diretório em Pxxx) serão gerados mesmo se a extensão .xxx porventura não exista no conjunto de artefatos do projeto. Isto permite, por exemplo, criar um diretório para arquivos contendo mensagens de erro.

Caso os dados estejam válidos, serão gerados os arquivos:

<Nome>.make conterá o *script* para MAKE. Este arquivo será criado no diretório base, ver linha de comando. <Nome> é o nome do construto a ser composto pelo MAKE, ver item NomeConstruto acima.

<Nome>.build conterá o *script* para o programa LINK ou para o gerente de bibliotecas LIB. Ele será criado no diretório associado com a extensão obj, ou na extensão default, caso obj não tenha sido definido. Se for definida a seção [SemLink], o arquivo não será gerado.

<Nome>.list conterá o relatório de composição do construto. Ele será criado no diretório base do construto. O arquivo de listagem pode ser dado, pela ordem, por parâmetro no arquivo de diretivas do construto, por parâmetro na linha de comando, ou por *default* conforme descrito neste item.

Procure sempre gerar um único arquivo contendo todas as mensagens de erro e advertência gerados no decorrer da reconstrução do construto. Isto pode ser conseguido redirecionando a saída dos compiladores para ser acrescentada (*append*) ao final de um arquivo de erros. Para evitar que as mensagens se tornem cumulativas no caso de reconstruções consecutivas, o arquivo erro deve sempre ser apagado ao iniciar a reconstrução. Esta é a forma *default* definida no arquivo <ArqPlat>. Caso seja seguida esta recomendação, será gerado ainda o arquivo:

<Nome>.err conterá a listagem de erros e advertências gerados pelos compiladores e ferramentas. Na realidade este arquivo é gerado sob controle das diretivas contidas no arquivo de diretivas da plataforma. Recomenda-se que todos os compiladores e ferramentas redirecionem a saída da console de modo a ser concatenada ao final do arquivo <Nome>.err sendo gerado.

```
[Diretorios]
   Nome              = MeuPrograma
   PathDefault       = ..\Ferramentas;..\Tabelas
   c                 = ..\Fontes
   h                 = ..\Fontes
   defstr            = ..\Tabelas;..\Definicoes
   inc               = ..\Tabelas;..\Definicoes
   tab               = ..\Tabelas;..\Definicoes
   err               = ..\Fontes
```

Figura 9. Exemplo de seção [Diretorios] em que são utilizadas tabelas

No exemplo da Figura 9 o diretório base do produto é c:\xxx\Composicao. Todos os demais diretórios serão relativos a este. Isto permite que, ao ativar o GMAKE, o diretório corrente seja:

- qualquer, neste caso deve ser fornecido o parâmetro de linha de comando /B indicando o diretório base onde se encontra o *script* make.
- o diretório base onde se encontra o *script* make.

Arquivos com nomes de extensão *diferentes* de .c, .h, .defstr, .inc, .tab e .err serão procurados no *path default*, ou seja, serão procurados primeiro no diretório ..\Ferramentas e depois, se não encontrados, no diretório ..\Tabelas. O programa executável será gerado no diretório base, a menos que um outro diretório seja indicado na seção [BuildInicio]. Caso seja gerado um arquivo .inc este será armazenado no diretório ..\Tabelas. Entretanto, ao ser procurado um arquivo .inc este será procurado

primeiro no diretório `..\Tabelas` e depois, se não encontrado, no diretório `..\Parametros`.

Os itens associando nomes de extensão a *paths* são opcionais. Devem ser utilizadas para estabelecer a estrutura padrão de diretórios. A política de procura por um arquivo é:

1. se o nome de um arquivo encontrado no código explorado contiver um nome de diretório, o arquivo será procurado neste diretório e somente nele.
2. se existe *path* associado com o nome de extensão do arquivo, a procura se dará neste *path*, e somente nele.
3. se não existe *path* associado com o nome de extensão do arquivo, mas tiver sido definido o `PathDefault`, o arquivo será procurado no `PathDefault` e somente nele
4. não existindo nenhum dos *paths* acima, o arquivo será procurado no diretório base definido ao ativar `GMAKE`.

4.2.2. Itens da seção `[Modulos]`

Cada linha de diretiva da seção `[Modulos]` tem o formato:

```
<NomeArq> <Modo>
```

Linhas em branco e comentários serão ignorados. Os itens têm o significado a seguir:

`<NomeArq>` é o nome de um arquivo a ser processado pelo `MAKE`. O `<NomeArq>` pode conter um nome de diretório, possivelmente relativo. Caso contenha, a procura pelo arquivo se dará exclusivamente neste diretório. O `<NomeArq>` pode ter nome de extensão. Caso não tenha, será utilizada a extensão *default*. A extensão *default* será sempre a primeira extensão definida na seção `[Comandos]` do arquivo de diretivas da plataforma `<ArqPlat>`. O nome de extensão é utilizado para selecionar a regra de geração definida no arquivo de diretivas da plataforma, e para selecionar o *path* onde se pode encontrar o arquivo, caso `<NomeArq>` não defina diretório. O `<NomeArq>` corresponde ao arquivo fonte (módulo de implementação) a ser processado. O nome do arquivo objeto a ser criado por `MAKE`, será obtido substituindo-se o nome de extensão contido em `<NomeArq>` (ou adicionado por *default*) pelo correspondente nome de extensão objetivo, tal como definido no arquivo de diretivas da plataforma. Ao criar a lista de módulos que compõem o construto coloque os nomes de componentes a serem gerados no início da lista e os nomes dos componentes a serem compilados no final.

`<Modo>` indica a forma de tratar o módulo. `<Modo>` é um *string* de um ou mais caracteres. São seguintes as formas:

`em branco` ou `/l` compila **e** liga (inclui o módulo objeto no *script* `.build`).

`/xxx` ou `-xxx` compila **e** liga. O texto `xxx` será incluído no comando que se encontra na primeira linha do conjunto de comandos do tipo de comando processado. Esta opção é utilizada tipicamente para incluir diretivas de definição de nomes no comando de compilação, por exemplo: `/D_DEBUG`.

`i` não compila, somente liga, i.e. inclui o nome do arquivo no *script* `.build`.

`n` ou `g` gera (compila) mas não liga. Esta opção é utilizada tipicamente para arquivos que serão processados por ferramentas específicas desenvolvidas pelo usuário. Em geral, essas ferramentas gerarão arquivos que serão utilizados ao compilar módulos específicos.

`v` nem compila nem liga, no entanto, verifica a existência do módulo, acusando erro caso não seja encontrado.

`outros` gera mensagem de erro.

Ao gerar as listas de dependências de um `<NomeArq>`, este será analisado, sendo registrados todas as inclusões no formato `#include "NomeInclusao"`, e, recursivamente, as inclusões contidas nestes arquivos de inclusão. Não serão examinados os arquivos de inclusão de módulos do compilador, ou seja, inclusões no formato `#include <NomeInclusao>`. Na seção `[Modulos]` aparecem somente os arquivos origem, mas não os arquivos de inclusão. Usualmente os arquivos origem são os módulos de implementação que constituem um determinado construto.

4.3. Formato do arquivo `<ArqPlat>`

O arquivo de diretivas da plataforma `<ArqPlat>` define os comandos específicos da plataforma de desenvolvimento e das linguagens de desenvolvimento utilizadas.

A composição geral do arquivo de diretivas da plataforma é idêntica à organização do arquivo de diretivas do produto. As seções de diretivas são:

`[Comandos]` Esta seção define os comandos de processamento dependendo do tipo do arquivo origem. Formato:

```
<extorg> => <extdest>
   Comando 1
   Comando 2
   ...
   Comando n
```

`<extorg>` é o nome de extensão de arquivo origem (*dependent*). O nome de extensão `<extorg>` deve ser redigido a partir do primeiro caractere da linha de diretivas. Pode existir somente uma diretiva de geração para cada extensão `<extorg>`. Para cada arquivo com extensão `<extorg>` encontrado na seção `[Modulos]`, será gerada uma diretiva de geração no *script* `.make`, utilizando os correspondentes *n* >= *1* comandos `Comando 1` a `Comando n`. ***Observação:*** `extorg` ***e*** `extdest` ***devem ser redigidos usando letras minúsculas.***

`<extdest>` é o nome de extensão de arquivo objetivo (*target*). Um mesmo `<extdest>` pode ser gerado a partir de vários diferentes `<extorg>`. Essencialmente as linhas do *script* de geração devem ser capazes de criar o arquivo `arq.extdest` a partir do arquivo `arq.extorg`.

`comando i` corresponde a uma linha de comando do sistema operacional. Estes comandos serão executados para gerar o arquivo objetivo

de extensão <extdest> a partir de arquivos origem de extensão <extorg>. Os comandos devem deixar o primeiro caractere da linha em branco[4], como é padrão para os comandos contidos em diretivas de MAKE. Entre os *n* comandos não deverão existir linhas em branco, no entanto poderão existir comentários (linhas iniciando com "#"), estes não figurarão no *script* gerado.

```
[Comandos]
c    =>    obj
   cl  $(O)  $(OPT)  /Fo$(Fobj)\  $(Fc)\$(@B).c   >> $(Ferr)\$(NOME).err

lista  =>  tab
   geratab  /L$(Ftab)\$(@B).lista  /T$(Ftab)\$(@B).tab  >> $(Ferr)\$(NOME).err

espstr =>  inc
   geratbdf  /E$(Ftab)\$(@B).espstr  /T$(Ftab)\$(@B).inc  >> $(Ferr)\$(NOME).err
```

Figura 10. Exemplo de seção [Comandos]. geratab e geratddf são ferramentas do usuário

O primeiro par de extensões define a extensão *default*. Esta será utilizada em todos os arquivos listados na seção [Modulos] e que não tenham nome de extensão.

[MacrosGerais] Esta seção define as macros MAKE a serem inseridas no arquivo *script* .make.

```
[MacrosGerais]

L       =
O       =
!IFDEF PRD
O       = /Ox
!ENDIF

OPT     = /c /J /W4 /nologo
INCLUDE = $(INCLUDE);$(PDEFAULT)
```

Figura 1112. Exemplo de seção [MacrosGerais]. São definidos os parâmetros de compilação.

[Limpa] Esta seção define os comandos de sistema operacional a serem ativados antes do primeiro módulo ser processado por MAKE. A seção é utilizada tipicamente para eliminar arquivos de mensagens de erro, e para estabelecer o contexto necessário para poder ativar os diversos processadores de linguagem utilizados.

```
[Limpa]
   IF EXIST $(Ferr)\$(NOME).err  DEL $(Ferr)\$(NOME).err
```

Figura 13. Exemplo de seção [Limpa]. Ilustra como eliminar o arquivo de mensagens de erro.

[DefaultLink] Esta seção define os comandos de sistema operacional padronizados do *script* .make necessárias para ativar a ligação. Estes comandos são incluídos no *script* .make caso não seja fornecido a seção [Link] nas diretivas de composição do construto.

[Fim] Esta seção indica o término de comandos. As linhas que seguem [Fim] são tratadas como comentários.

[4] Sugere-se utilizar uma endentação de 3 caracteres.

```
[DefaultLink]
    cd $(Fobj)
    LINK $(L) @$(NOME).build >> $(Ferr)\$(NOME).err
```

Figura 14. Exemplo de seção `[DefaultLink]`. A mudança de diretório assegura que o LINK operará sobre os módulos objetos e a diretiva `.build` do construto.

5. Esqueleto do arquivo `<Nome>.make` gerado

No texto a seguir os elementos em **negrito** serão substituídos pelos correspondentes textos de diretivas

```
############################################################
##### Script de MAKE para o produto: <NomeConstruto>
##### Gerado a partir de: <ArqConst>
############################################################

### Nomes globais

NOME     = <NomeProduto>

### Nomes de paths

PDEFAULT = <PathDefault>
Pppp     = <Path associado com a extensão ppp>

### Nomes de diretórios para geração

FDEFAULT = <FolderDefault>
Fppp     = primeiro diretório em Pppp, para todos os ppp para os
           quais existe uma regra de conversão <extorg> => <extdest>.
           Deve ser utilizado no comando de compilação.

### Macros iniciais do construto

<Diretivas [MacrosAntes]>

### Macros gerais da plataforma

<Diretivas [MacrosGerais]>

### Macros finais do construto

<Diretivas [MacrosApos]>

### Regras de geração

all : limpa  \
   <Diretivas [ArtefatosAntes]> \
   <lista dos componentes a gerar > \
   <Diretivas [ArtefatosApos]> \
   Construto

#### Limpar arquivos

limpa :
   <Diretivas [Limpa]>
   <Diretivas [LimpaMais]>

### Dependências dos arquivos a gerar

$(FOBJ)\<modulo1>.<extdest1>:  $(F<extorg1>)\<modulo1>.<extorg1>  \
   <includes requeridos pelo modulo1, gerados por gmake>
<Diretivas de compilação associadas a <extorg1>>

...

### Dependências adicionais do produto
```

```
<Diretivas [GeraMais]>

### Composição final

Construto:  \
   $(FOBJ)\<modulo1>.<extdest1> $(FOBJ)\<Modulo2>.<extdest2> ... \
   <Diretivas [Link], ou [SemLink]> \

################################################
# Fim de script MAKE para o construto <NomeConstruto>
################################################
```

6. Esqueleto do arquivo <Nome>.BUILD gerado para [LINK]

Build é utilizado nas novas versões de Visual C/C++ e outros sistemas de compilação.

```
<Diretivas [BuildInicio]>
<modulo1 ligável>  <modulo2 ligável>  &
<modulo3 ligável>  ...
<Diretivas [BuildFim]>
```

7. Conteúdo do arquivo <prod>.LIST gerado

```
Nome arquivo de diretivas do produto:
Nome arquivo de diretivas gerais:

Nome do produto
Nome do arquivo .MAK:
Nome do arquivo .LNK (ou .LIB):
Nome do aquivo listagem:
Diretório corrente:
Diretório do produto:
Diretório dos objetos:

Path default:
Path da extensão ppp1:
Path da extensão ppp2:
...

Estatísticas dos arquivos utilizados pelo produto

Arquivo1
Número de linhas de código:
   Número de linhas de comentário:
   Número de linhas em branco:
   Número total de linhas do arquivo:
Lista de arquivos que este arquivo inclui
Lista de arquivos nos quais este arquivo é incluido

Arquivo2
Número de linhas de código:
   Número de linhas de comentário:
   Número de linhas em branco:
   Número total de linhas do arquivo:
Lista de arquivos que este arquivo inclui
Lista de arquivos nos quais este arquivo é incluido

...

Estatísticas por nome de extensão

Extensão xxx1
   Número de arquivos processados com extensão xxx1:
   Número de linhas de código extensão xxx1:
   Número de linhas de comentário extensão xxx1:
   Número de linhas em branco extensão xxx1:
```

```
   Número total de linhas de arquivos com extensão xxx1

Extensão xxx2
   ...

Estatística final
   Total de arquivos processados
   Número de linhas de código total:
   Número de linhas de comentário total:
   Número de linhas em branco total:
   Número de linhas total:
```

São consideradas linhas de comentários:

- linhas contendo `/*` ou linhas continuação de comentário, até a linha que contenha `*/`. Não são contadas como linhas de comentários, linhas contendo comentários com menos do que 7 caracteres excluindo `/*` e `*/`. Desta forma comentários de término de estrutura não serão contados.
- linhas contendo `//`, desde que sejam seguidos por 7 ou mais caracteres. Desta forma comentários de término de estrutura não serão contados.
- linhas contendo `*` na primeira coluna
- cabe salientar que uma linha do arquivo contendo código e comentário longo pode ser contada duas ou mais vezes.

São consideradas linhas em branco:

- linhas contendo somente brancos ou tabulações

São consideradas linhas de código

- linhas contendo caracteres fora do domínio de comentários. Note que um comando longo pode ser formado por diversas linhas. Optamos por esta forma de contagem pois comandos longos são muito mais complexos do que comandos curtos. Na média o número de linhas contado assim deve corresponder mais exatamente ao esforço requerido para redigir o programa, embora seja sensível ao padrão de formatação utilizado pelo programador.

Apêndice A. Exemplo de diretivas de plataforma GMAKE.PARM

```
# Diretivas para plataforma de compilacao MS visual c/c++ 6 ou mais recente

[Comandos]
c  =>  obj
   cl $(O) $(OPT) /Fo$(Fobj)\  $(Fc)\$(@B).c  >> $(Ferr)\$(NOME).err

[MacrosGerais]
# Use O  para compilar m¢dulo normal
#     OD para compilar m¢dulo para o debugger
#     L  para ligar normal
#     LD para ligar com debugger
#
O       = /MLd /Gs
OD      = /MLd /Gs /Zi
L       =
LD      = /DEBUG /DEBUGTYPE:CV

CDB     = /D_DEBUG
OPT     = /c /J /W4 /EHsc /G5 /GA /Gy /GF /GR /Zl /Zp /nologo
R       = /r

!IFDEF PRD
O       = /ML /Ox
CDB     =
!ENDIF

[Limpa]
   IF EXIST $(Ferr)\$(NOME).err  DEL $(Ferr)\$(NOME).err
   SET CL=$(OPT)

[DefaultLink]
   SET CL=
   cd $(Fobj)
   LINK $(L) @$(NOME).build  >> $(Ferr)\$(NOME).err

[Fim]
```

Apêndice B. Exemplo de diretivas `.COMP` de um produto simples

```
# Composição do programa exemplo de teste automatizado
# Utiliza o arcabouço simplificado

[Diretorios]

Nome        =  ExemploSimples
err         =  .

[MacrosAntes]
O      = /Zi /Od
L      = /DEBUG /DEBUGTYPE:CV

[Modulos]
Princ
Testegen
Testespc
Arvore

[BuildInicio]
/OUT:ExemploSimples.exe
/COMMENT:"Exemplo de teste automatizado. (c) LES/DI/PUC-Rio 2004"
/INCREMENTAL:NO
/MACHINE:IX86

[Fim]
```

Apêndice C. Exemplo de diretivas .COMP

```
# Composição do programa teste automatizado do arcabouço de teste automatizado para C
# usando a biblioteca estática do arcabouço

[Diretorios]

Nome        =  TesteArcaboucoLib
PathDefault =  ..\Fontes
err         =  ..\Produto
obj         =  ..\Objetos
c           =  ..\Fontes
h           =  ..\Fontes
def         =  ..\Tabelas
tab         =  ..\Tabelas
inc         =  ..\Tabelas

[Modulos]
Arvore
TestEspc

[BuildInicio]
/OUT:..\produto\TesteArcaboucoLib.exe
/INCREMENTAL:NO
/MACHINE:IX86

[BuildFim]
ArcaboucoTeste.lib

[Fim]
```

Apêndice D. Exemplo de arquivo `.COMP` para gerar uma biblioteca

```
# Composição da biblioteca estática do arcabouço de teste automatizado para C

[Diretorios]

Nome        =  TesteLib
PathDefault =  ..\Fontes
err         =  ..\Produto
obj         =  ..\Objetos
c           =  ..\Fontes
h           =  ..\Fontes
def         =  ..\Tabelas
tab         =  ..\Tabelas
inc         =  ..\Tabelas

[Modulos]
Princ
Testegen
GenTest
Cntespac
Intespac
Counters
IntCnt

[Link]
    cd $(Fobj)
    LIB $(L) @$(NOME).build >> $(Ferr)\$(NOME).err

[BuildInicio]
/OUT:ArcaboucoTeste.lib
/MACHINE:IX86

[Fim]
```

Apêndice E Exemplo de arquivo .COMP usando ferramentas do usuário

```
# Composição do teste de uso de tabela de string

[Diretorios]

Nome        =  ExemploTabela
PathDefault =  .
err         =  .
h           =  .
c           =  .
inc         =  .
espstr      =  .
tab         =  .
lista       =  .


[Modulos]
TabelaString.lista        g
TesteTabelaString.espstr  g
Tabstr
TesteTbs

[BuildInicio]
/OUT:ExemploTabela.exe
/INCREMENTAL:NO
/MACHINE:IX86


[BuildFim]
ArcaboucoTeste.lib

[Fim]
```

Na seção [Comandos] do arquivo de diretivas da plataforma devem ser acrescentadas as linhas:

```
lista  =>  tab
   geratab   /L$(Ftab)\$(@B).lista   /T$(Ftab)\$(@B).tab  >> $(Ferr)\$(NOME).err

espstr =>  inc
   geratbdf  /E$(Ftab)\$(@B).espstr /T$(Ftab)\$(@B).inc   >> $(Ferr)\$(NOME).err
```

Apêndice F. Exemplo de arquivo .BAT para ativar o MAKE

```
@ECHO  OFF
REM  Compila programa C/C++ usando nmake %1.make  do conjunto MS Visual c/c++
REM  Sintaxe:  faz  nome-do-arquivo.mak  [o]
REM                   opção  o : compila otimizado. Todos os obj devem
REM                              ter sido destruidos antes
REM  Precisa ajustar o   call vsvars32.bat   para estar de acordo com a
REM                      instalacao de C/C++ na plataforma em uso. Procure
REM                      o arquivo vsvars32.bat (ou o vcvars32.bat)
REM                      e utilize o correspondente endereço
REM  Pode necessitar o ajuste da secao :contin para que o NotePad abra
REM                      corretamente o arquivo de mensagens de erro geradas
REM                      ao compilar

pushd  .

if ""=="%1" goto erro

if ""=="%VCENV%" call vsvars32.bat
set VCENV="set"
del *.err

if "o"=="%2" goto otimz
if "O"=="%2" goto otimz

nmake /F..\composicao\%1.make
goto contin

:otimz
nmake /F..\composicao\%1.make "PRD="

REM substitua notepad pelo seu editor preferido
:contin
notepad ..\produto\%1.err

goto sai

:erro
echo Falta nome do arquivo a compilar

:sai
popd
```