PCS 3216
Sistemas de Programação

Aula 12

Ligação, Ligadores, Overlays

Projeto de um Relocador-Ligador

LIGAÇÃO E LIGADORES

Introdução

- Nesta aula o foco do estudo é dirigido aos softwares responsáveis por viabilizar o desenvolvimento **modular** dos programas, em um sistema de programação.
- Como vimos na aula anterior, a modularização é obtida quando se dispõe de algum mecanismo através do qual programas independentemente desenvolvidos possam **referenciar-se mutuamente** pela citação dos **nomes simbólicos** declarados nos diversos módulos.
- Vimos ainda que os **ligadores** são os responsáveis pela resolução dessas referências simbólicas entre os módulos, e que tal operação se chama **ligação**.

- A operação de ***ligação***, que efetua a associação de endereços aos nomes simbólicos utilizados na comunicação entre módulos, é também conhecida na literatura como ***linking***, ***binding***.
- Os programas de sistema utilizados para realizar esta tarefa denominam-se ***ligadores, linkers***, ***binders***.
- Esses programas assumem diversas formas, conforme a aplicação a que se destinam e os métodos empregados para a sua realização.

Ligadores-relocadores

- Podem apresentar-se como programas muito simples, encarregados apenas de **ligar** os módulos entre si, e **relocá-los**, obtendo diretamente um programa absoluto executável.
- Conforme o caso, incorporam também as funções do **loader**, carregando diretamente na memória para execução o programa absoluto produzido.
- Tantas variantes dos ***ligadores-relocadores*** recebem também nomes diversos: ***relocadores-ligadores, relocating loaders, linking loaders***.

- As diversas funcionalidades citadas podem realizar-se através de programas que operam separadamente:
  - um ***ligador*** (***linker, binder***), encarregado apenas da resolução dos endereços simbólicos entre módulos, produzindo um programa objeto relocável sem referências simbólicas.
  - um ***alocador de memória*** (***locator***), encarregado de distribuir, na memória disponível, os módulos da melhor maneira, para efeito de associação de endereços absolutos aos mesmos. Em geral esta função é do sistema operacional.
  - um ***relocador*** (***relocating loader***), encarregado da resolução do endereçamento relativo apresentado pelas instruções de referência à memória, especialmente aquelas entre módulos, produzindo um programa objeto absoluto.
  - Um ***carregador (loader),*** encarregado de carregar na memória para execução o programa absoluto produzido pelo relocador.

- Neste esquema, e possível a operação parcial do processo completo de preparação de um programa para a execução, permitindo, independentemente:
  - a **ligação de alguns módulos** entre si,
  - a **relocação de alguns módulos** e ainda
  - a **geração de código absoluto** para alguns módulos.
- Isto muito flexibiliza o sistema, razão que justifica a adoção deste método em diversos sistemas de desenvolvimento modernos.

OVERLAYS

Overlays

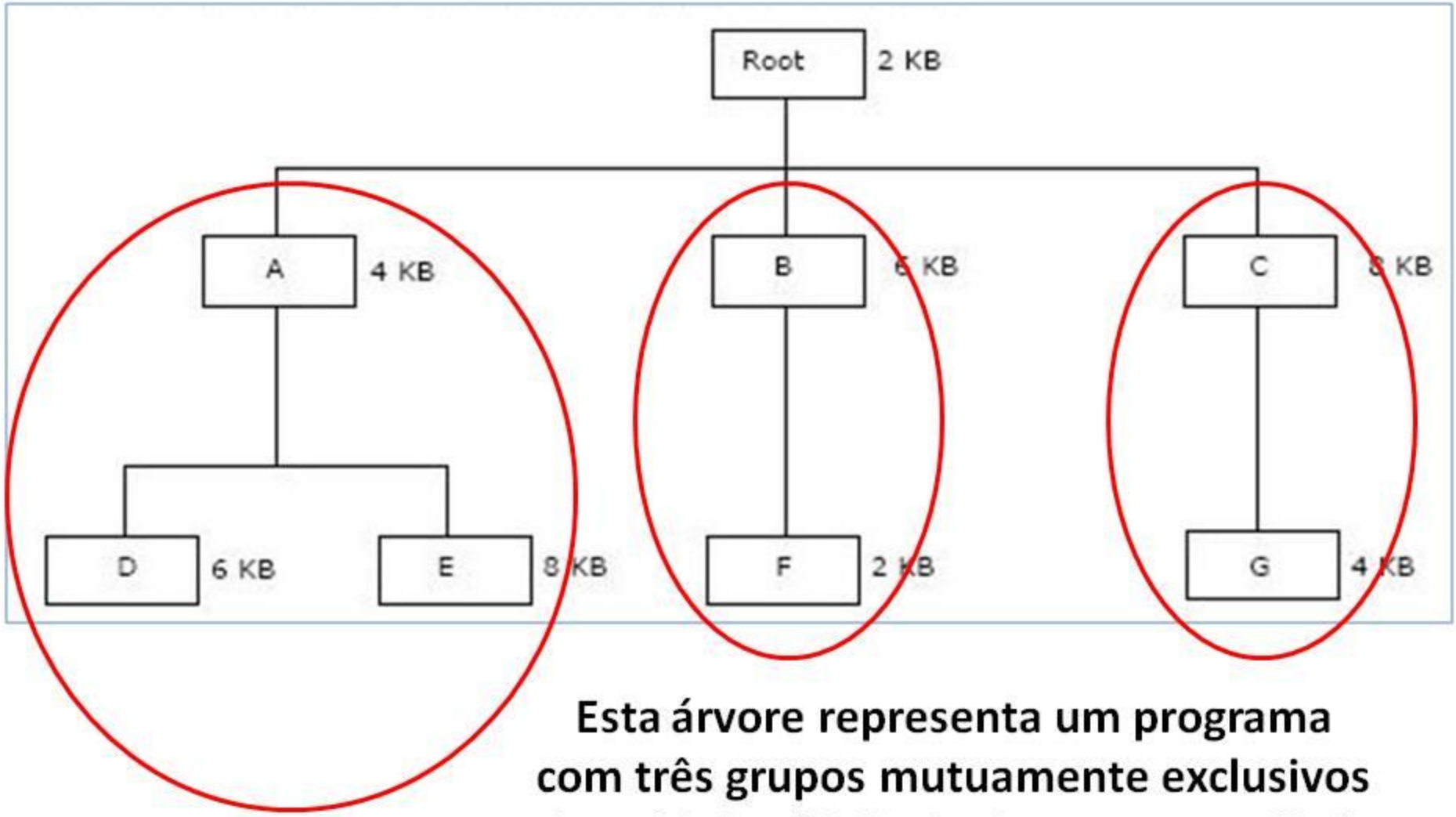
- Para alguns outros sistemas, pode ser utilizada a estratégia conhecida como **sistema de *overlays***:
- Trata-se de uma antiga mas eficaz **técnica manual** de alocação, que viabiliza executar programas grandes, mesmo com disponibilidade menor de memória física.
  - uma **área de memória** é reservada para **alojar áreas** de dados e de programas **que sejam comuns a vários módulos**
  - uma **outra (que se chama área de *overlay*)** é preferencialmente destinada a abrigar **partes mutuamente exclusivas** do programa, ou seja, aquelas das quais nunca é necessária a presença física simultânea na memória durante a execução do programa.
  - Dessa forma, partes mutuamente exclusivas do programa nunca convivem, liberando assim o máximo de área na memória física para que possa ser ocupada por outras partes que, de outra maneira, não poderiam ser executadas por falta de espaço.

Observações sobre *Overlays*

- A clássica técnica do ***Overlay*** tem como meta **permitir a execução de programas maiores que a memória** disponível.
- É do **programador** a responsabilidade de **particionar o seu programa** adequadamente, de forma que sejam **minimizadas as mudanças de contexto** durante a execução.
- Em cada instante, devem permanecer **residentes os dados e as instruções necessários** à execução do programa na ocasião.
- Pode não ser imediato projetar a estrutura de *overlays* para os programas, pois **o sistema operacional dificilmente dá suporte** automático para essa técnica.

- É usual que os programas que empregam esta técnica sejam manualmente estruturados como uma **árvore de execução**.
- **Em cada instante** da execução, precisam obrigatoriamente estar residentes na memória apenas aqueles **módulos** que correspondam aos nós **dessa árvore presentes no caminho** que leva do módulo em execução até a raiz.
- Como o sistema não entra no mérito da forma como o programa é particionado, nem da dinâmica de substituição dos seus *overlays*, é **responsabilidade exclusiva do usuário** projetar, estruturar e garantir que em tempo de execução todas as condições operacionais do programa sejam devidamente respeitadas.

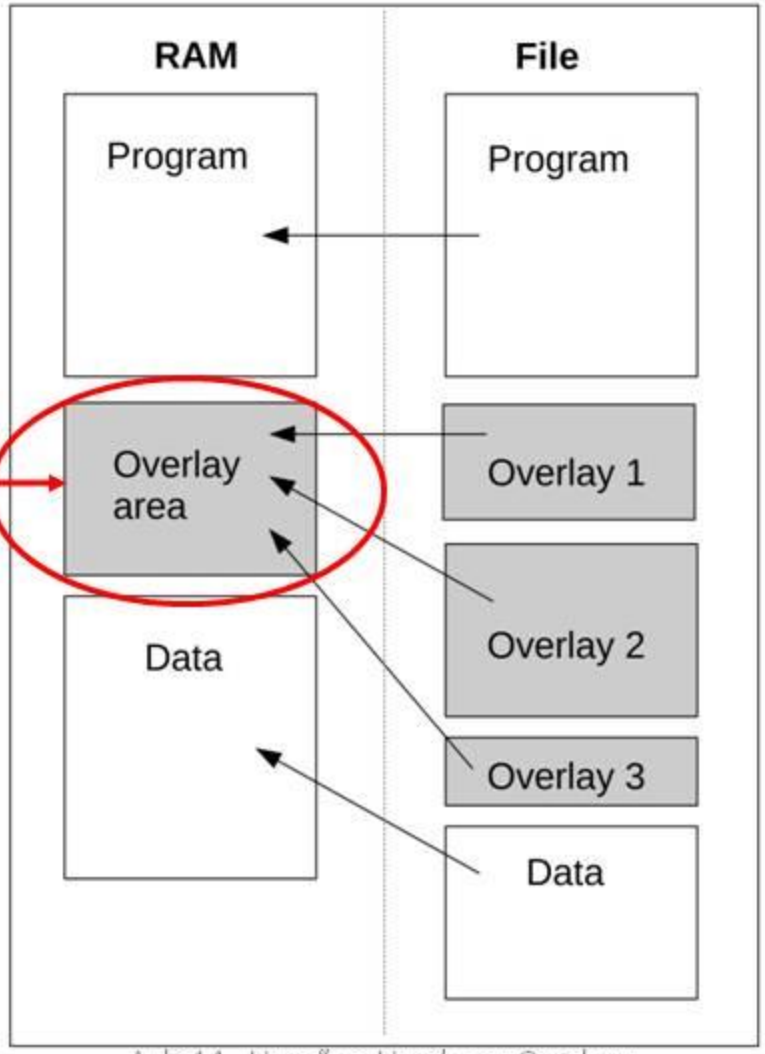
Uma árvore de overlays



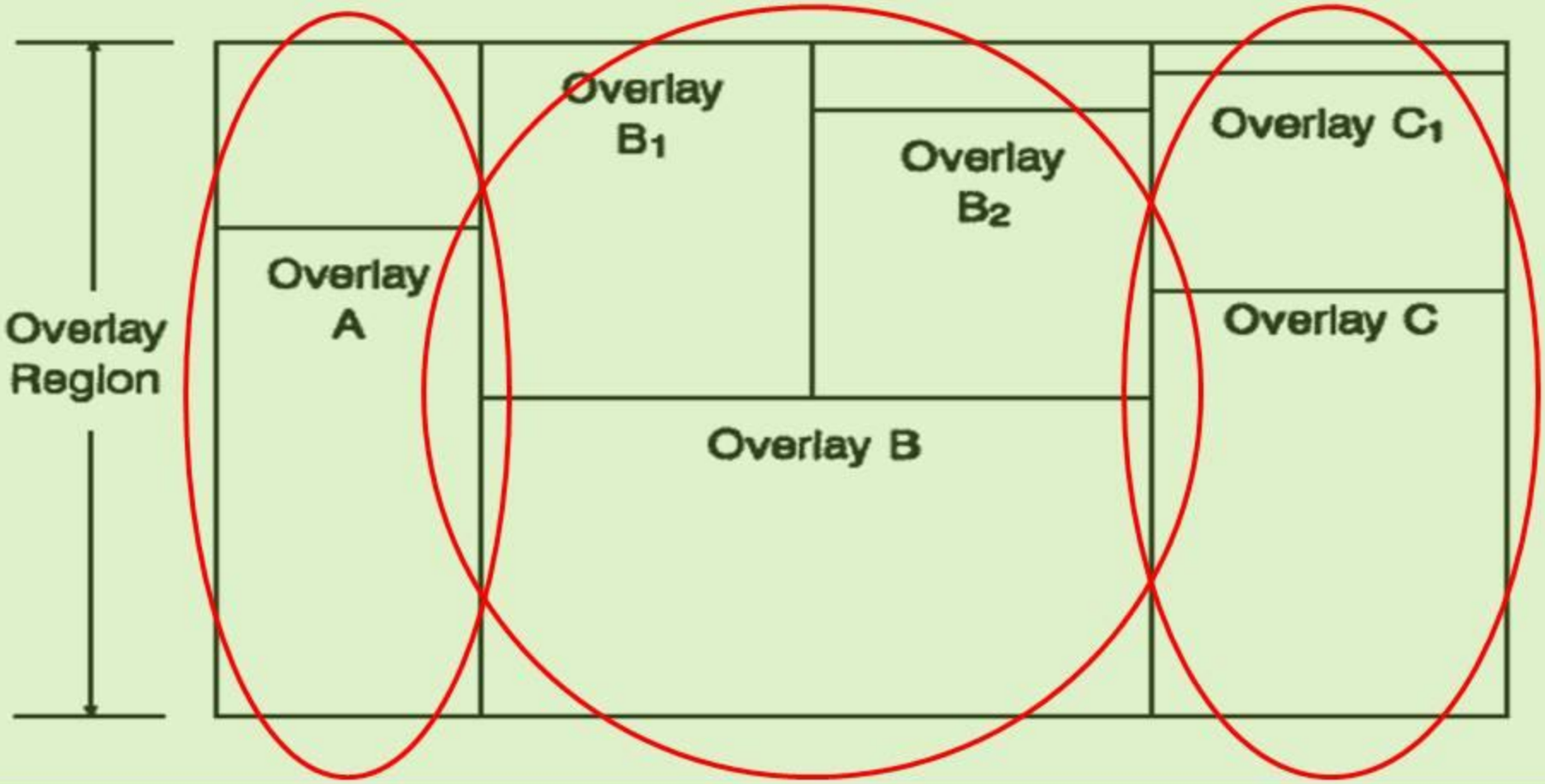
Esta árvore representa um programa com três grupos mutuamente exclusivos de módulos (delimitados em vermelho).

Operação de um programa com uma área compartilhada por 3 overlays

Os overlays 1,2 e 3 ocupam, *um de cada vez*, esta mesma área da memória RAM.



Programa com hierarquia de overlays



Os overlays A,B e C ocupam, *um de cada vez*, a mesma "Overlay Region" da memória RAM. Note que, por sua vez, B comanda sua própria área de overlay particular, na qual se revezam dois overlays menores (B_1 e B_2).

- Para o acerto e a resolução dos endereços relativos contidos nos ***overlays***, os ligadores, relocadores e alocadores deverão executar algoritmos adequados, que levem em conta as características das partes do programa que desempenhem o papel de ***overlays***.
- O uso da técnica de ***overlay*** permite ao usuário utilizar o computador para executar um programa mesmo que a área de memória disponível não seja suficiente para comportar totalmente o programa de uma só vez.

VIRTUALIZAÇÃO DE MEMÓRIA

Conceito

- O uso dessa técnica representa, na evolução dos sistemas de programação, um grande passo em direção aos sistemas modernos de computação.
- Convém lembrar que historicamente os ***overlays*** significaram uma decisiva vitória contra as limitações do espaço de armazenamento na memória principal.
- Os ***overlays*** acabaram representando, na história, a principal precursora da técnica da ***virtualização de memória***, amplamente empregada hoje nos sistemas modernos, e que é tema de estudo em outro contexto.

Interrupção de falta de ligação

- Alguns computadores dispõem de um suporte de hardware que permite **adiar a época da ligação** dos módulos até o instante exato em que eles venham a se fazer necessários para a execução do programa.
- Este suporte manifesta-se na disponibilidade de uma **interrupção** especial de ***falta de ligação***.
- A disponibilidade dessa interrupção na máquina permite iniciar a execução de programas mesmo que nem todos os seus módulos já se encontrem totalmente ligados e suas referências, resolvidas.

Detalhes da operação

- No instante em que é feita uma referência a um módulo ainda não ligado:
  - Ocorre uma **interrupção**.
  - O hardware provoca a execução, em modo supervisor, logo dentro do sistema operacional, de um programa de tratamento dessa interrupção.
  - Nesse programa de tratamento, **desencadeia-se a execução** de uma versão especial do programa **relocador, com a finalidade de** resolver seus endereços simbólicos e relativos.
  - Ao retornar da interrupção, o programa fica, assim, pronto para ser executado pelo hardware.
- **Notar a semelhança (intencional)** entre o hardware de interrupção e os mecanismos implementados por **um motor de eventos**.

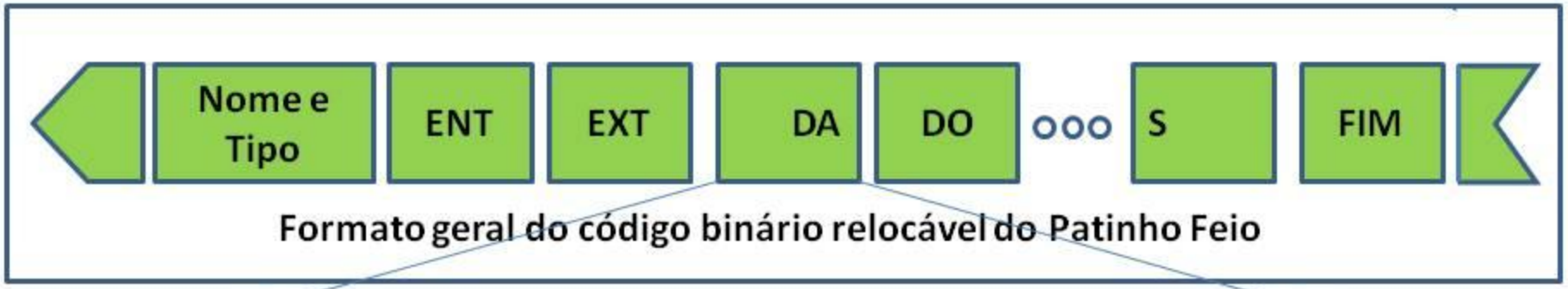
- O esquema que acabou de ser descrito é denominado ***ligação e relocação dinâmica***.
- Uma de suas vantagens é a de possibilitar ao usuário um tempo de relocação e de ligação muito menor durante a preparação, para a execução, de programas formados de muitos módulos, quando muitos deles são solicitados raramente.
- Outra é a agilização da preparação do programa para a execução, na época de desenvolvimento de programas, devido à não obrigatoriedade da ligação e relocação completa de todos os módulos, e também dada a frequência com que as operações de ligação e relocação precisam ser feitas nessa ocasião.

- A análise pormenorizada dessas técnicas, voltadas à **virtualização de recursos** da máquina, costuma ser feito juntamente com os aprofundamentos motivados pelo estudo do funcionamento dos programas que compõem os sistemas operacionais, tema estudado em outra disciplina.

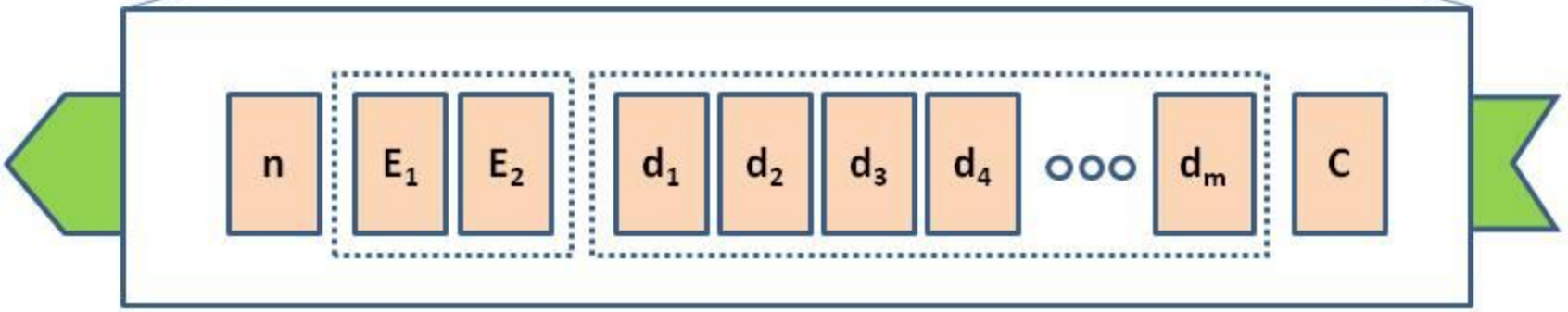
O material a seguir reflete a íntegra de uma documentação histórica do projeto de um relocador-ligador desenvolvido para o computador Patinho Feio por volta do ano de 1972.

FORMATO DO CÓDIGO RELOCÁVEL

FORMATO GERAL



Formato geral do código binário relocável do Patinho Feio



Formato geral dos diversos blocos do código relocável (qualquer tipo)

n = m-1 (Obs.: isto sinaliza que é inadequado o uso de blocos vazios)

m = número de dados da sequência de dados d_1 ... d_m

C = checksum (complemento de 2 da somatória (módulo 256) dos bytes anteriores)
(Obs.: a soma de todos os bytes do bloco, incluindo n e o checksum, deve ser nula)

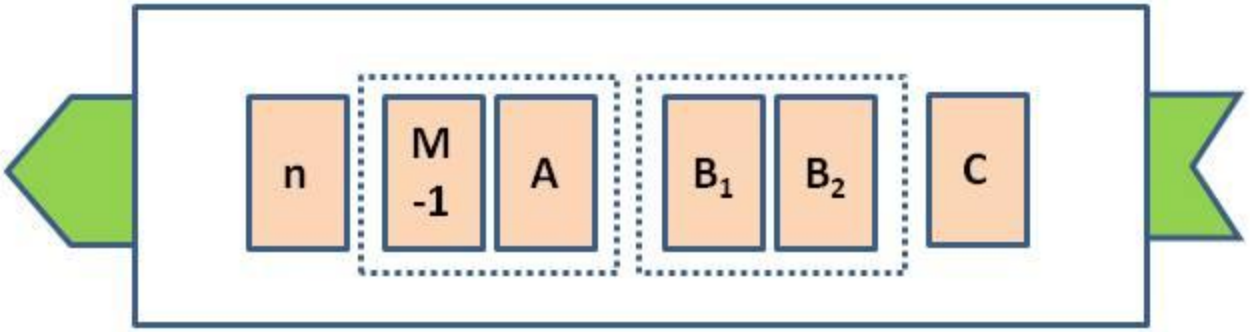
Ver, na lista a seguir, os significados dos símbolos utilizados nas figuras dos slides seguintes.

Códigos usados nos blocos

n	Número de palavras de 8 bits que vem a seguir, incluindo o checksum (-1 indica que é um bloco de Nome/Tipo)
A	0 => programa principal; ≠0 => subprograma
B_1,B_2	Nome do programa/subprograma empacotado em dois bytes (usando a rotina PACK)
C	Complemento de 2 da soma (módulo 256) dos bytes do bloco
D	Número de entry points
E_i,E_{i+1}	Nome do ((i+1)/2)-ésimo entry point (i=1,3,5...2*D+1)
e_i,e_{i+1}	Endereço relocável do ((i+1)/2)-ésimo entry point
F	Número de externals
G_i,G_{i+1}	Número do ((i+1)/2)-ésimo external (i=1,3,5...2*F+1)
g_i,g_{i+1}	N°. pelo qual ele vai ser referenciado no programa (é o end. relocável na tab. de subrotinas do ligador-relocador)
H_i	0 => abs., de 1 byte; 1 => relocável, de 2 bytes; -1 => relocável, de 2 bytes em rel. à tab. de subr. do ligador-relocador
I_1,I_2	Origem relocável do conjunto de bytes que vêm a seguir
J_i,K_i	i-ésimo byte do código: se H_i =0 => J_i = dado absoluto de 1 byte, K_i=0 se H_i =1 => J_i = primeiro byte relocável, K_i= segundo byte relocável
L_1,L_2	Endereço de execução do programa principal, ou então =0, se for subrotina.
M	Sequência dos blocos: -1 nome/tipo (um); -2 ENT (um) ; -3 EXT (um); -4 dados (o suficiente); -5 FIM (um);

Entre um bloco e outro, são incluídos 4 bytes nulos para permitir a localização do início do bloco por parte do usuário.

Bloco de NOME e TIPO



Formato geral do blocos de *Nome e Tipo*

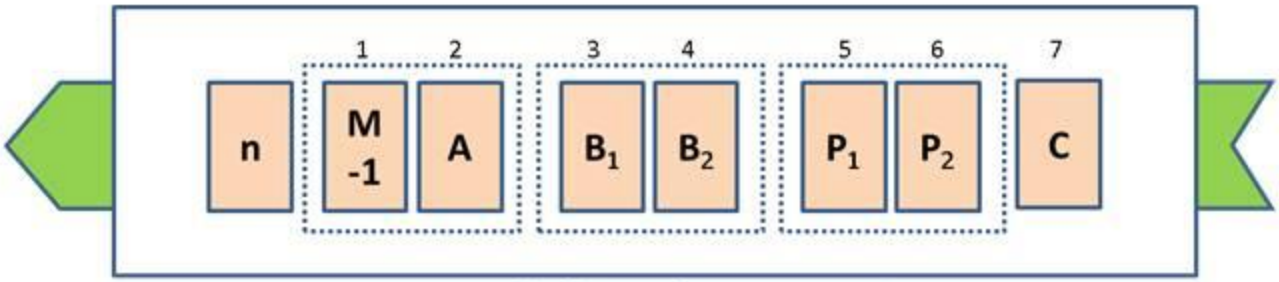
n = número de bytes seguintes no bloco (no caso acima, n=5)

NOME E TIPO => um ou nenhum bloco:

- se o código tiver um bloco de NOME, é relocável, ∴ OK.
- se não tiver, será absoluto, ∴ mensagem de erro.

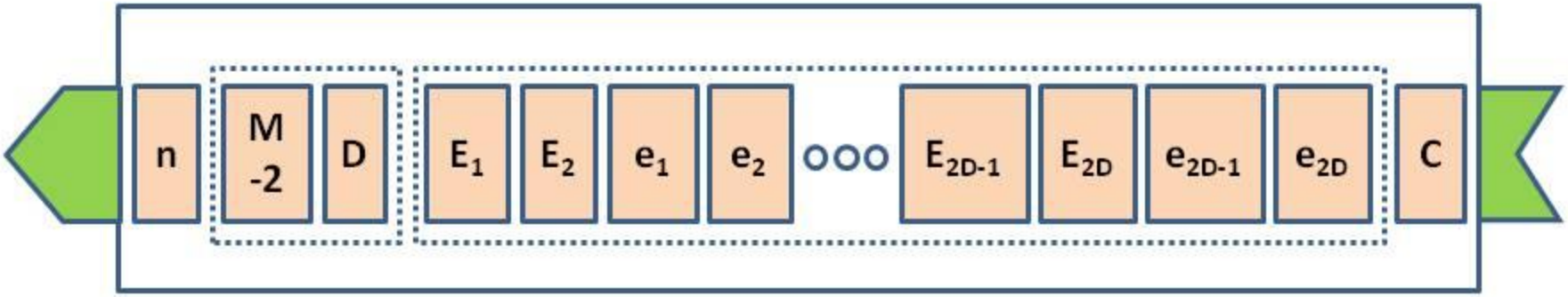
É possível incluir no bloco "NOME" uma informação sobre o comprimento total do programa (número de bytes ocupado pelo programa ou subrotina).

Neste caso, "NOME" ficaria conforme a figura seguinte, onde (P1,P2) contém [em 2 bytes] o número de bytes ocupado pelo programa ou subrotina.



Formato do bloco de *Nome e Tipo*,
[incluindo o comprimento do programa] (n = 7; m = 1)

Bloco ENT

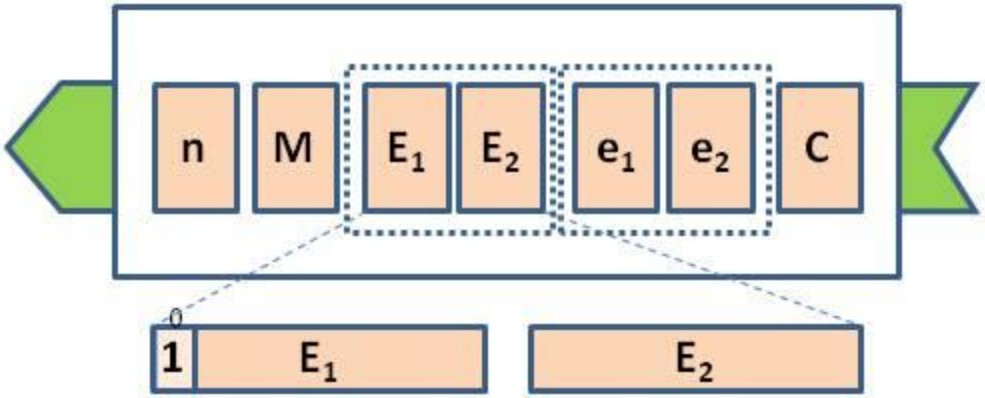


Formato geral do bloco ENT

ENT => duas hipóteses:

- Ou usar como está na figura (um só bloco, para todos os ENTRY (mais trabalhoso)).
- Ou fazer um bloco de ENT para cada declaração ENT no programa (mais fácil) .

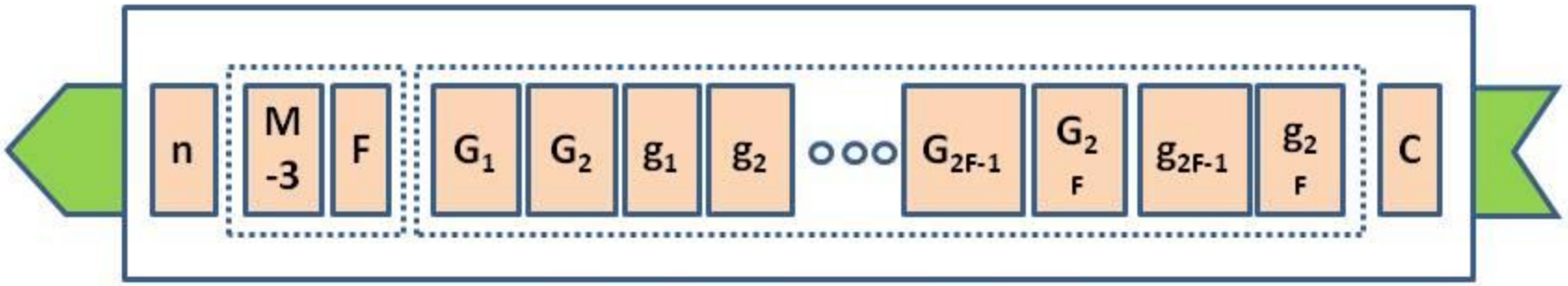
Neste caso, o ligador-relocador deve garantir que a sequência dos blocos esteja correta, e um bloco de ENT ficaria com o formato seguinte:



Formato alternativo do bloco ENT:

n = 6; m = −2

Bloco EXT

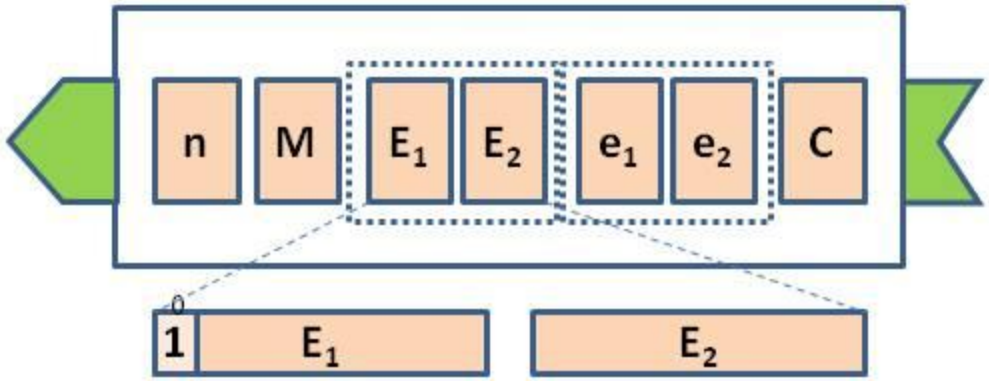


Formato geral do bloco EXT

EXT => duas hipóteses:

- Ou usar como está na figura (um só bloco, para todos os EXT (mais trabalhoso)).
- Ou fazer um bloco de EXT para cada declaração EXT no programa (mais fácil) .

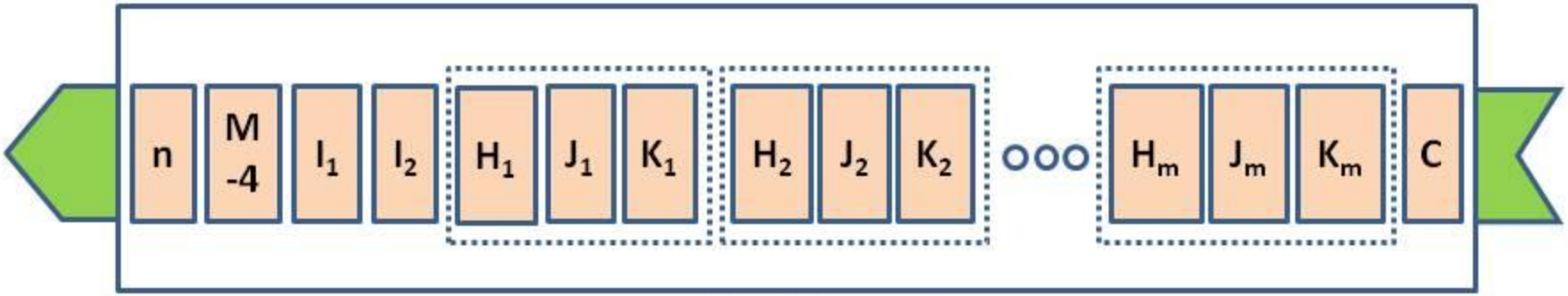
Neste caso, o ligador-relocador deve garantir que a sequência dos blocos esteja correta, e um bloco de EXT ficaria com o formato seguinte:



Formato alternativo do bloco EXT:

n = 6; m = −3

Bloco de DADOS



Formato geral dos blocos de DADOS

n = m-1

DADOS

-Usar tantos blocos quantos forem necessários para comportar todo o código do programa.

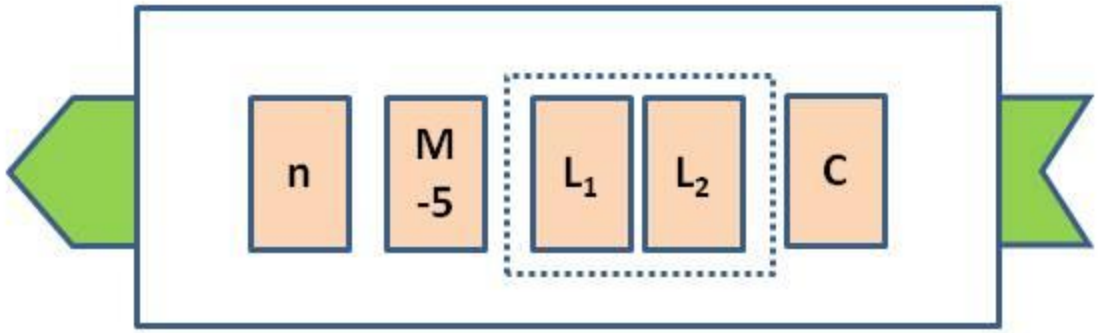
-[Para simplificar o processamento deste tipo de blocos, e também para tornar menos difícil a tarefa de decifrar o código relocável] em um bloco devem comparecer [apenas] informações completas.

- Por isso, não convém separar uma instrução em dois blocos diferentes.

Def./Indef. **Def.** => endereço de A

Indef. => endereço da subrotina chamadora de A

Bloco FIM



Formato geral do bloco de FIM

FIM

-É único para cada rotina ou programa principal, separadamente compilados.

COMO O LIGADOR-RELOCADOR DEVE FUNCIONAR

Como funciona o Ligador-relocador

1. Ler um bloco na fita. Testar na leitura o checksum.
   Se não bater => Mensagem de erro, aguardar releitura do bloco.
   Testar se o 1º bloco é do tipo "NOME".
   Se for => testar A para descobrir se é programa principal ou subrotina.
   Se for subrotina => erro (o 1º código deve ser do programa principal).
   Se for prog. principal => OK => Guardar nome comprimento numa tabela
   Testar se (CR (Constante de relocação) + comprimento do programa) invade a área protegida. Guardar a soma em uma variável auxiliar CR1.
   Se invadir => mensagem de erro (overflow de memória) (irrecuperável)
   Se não invadir => OK => continua o processo.
2. Ler outro bloco na fita. Deve ser um bloco EXT ou DADOS.
   Se EXT => Vá para 3); Se DADOS => vá para 4);
   Se não for => Mensagem de erro (irrecuperável)

3. Se for EXT => Testa se já consta na tabela.
   Se não constar na tabela, então guardar na tabela de subrotinas o nome da subrotina chamada, reservando mais 6 palavras para montar a subrotina chamadora.

Observação:

A solução adotada como mecanismo de ligação é bastante simplista:
Quando não se dispõe de um endereço conhecido para o símbolo externo, cria-se um endereço conhecido na tabela de subrotinas, e usa-se este no lugar do endereço desconhecido.
Quando se tornar conhecido este endereço, inclui-se na tabela de subrotinas uma instrução de chamada para a subrotina propriamente dita, na área artificialmente criada para essa operação.
O resultado é a execução de algumas instruções a mais, tornando mais lento o processo. Dificulta-se também o mecanismo de passagem de parâmetros.

O formato dessa informação será como está mostrado na tabela a seguir:

Informações sobre chamadas externas

1 byte	Número da referência EXT	Para cada rotina, o número do EXT da mesma subrotina pode variar. Este byte indica qual é o número do EXT na subrotina ou no programa corrente.
2 bytes	Nome da subrotina	
2 bytes	X PLA *-*	Este endereço é calculável sabendo onde começa a tabela
2 bytes	PUG *-*	Resevado para PUG (Subrotina), que vai ser montado aqui quando a subrotina for carregada na memória (ou seja, quando for lido o correspondente bloco ENT na fita objeto relocável das subrotinas)
2 bytes	PLA X	Retorno ao programa chamador.

- Obs.: O "nome da subrotina" deverá estar fisicamente em outra área de memória para liberar o máximo espaço possível para o programa. Os outros seis bytes finais ficarão residentes na memória durante a execução do programa.
- Cada subrotina exige, para o link, de 6 bytes na tabela de subrotinas, e mais 3 bytes para uso do ligador-relocador (residente na memória durante a execução do programa, inclusive)
- Lido um bloco de DADOS, não pode aparecer nenhum novo EXT até que um bloco FIM seja detectado, e que um novo bloco de NOME seja eventualmente lido.

4. Se o bloco lido for um bloco de DADOS, o ligador-relocador só poderá aceitar em seguida outro bloco de DADOS ou um bloco de FIM, desde que o comprimento não exceda o que foi declarado no bloco de NOME, caso em que deverá se emitida uma mensagem de erro (*).

5. Processamento do bloco de DADOS:
(a) Somando (I_1 I_2) à constante de relocação, teremos o endereço absoluto onde deverá ser armazenado o primeiro dado (guardar este endereço em CI (contador de instruções))
(b) obter H_i para saber se o material que vem em seguida é absoluto ou relocável.
Se for absoluto => guardar J_i no endereço apontado em CI, e somar 1 em CI. Ignorar K_i neste caso.
Se for relocável => somar (J_i,K_i) com a constante de relocação. Testar se o código foi alterado (em caso afirmativo => erro de overflow de memória; caso contrário, guardar o resultado em (CI) e (CI)+1, e somar 2 em CI.
Se for relocável em relação à tabela de subrotinas => gerar link correspondente.
(c) Se não terminaram os dados do bloco, voltar a (b);
se terminaram, ler novo bloco.
Se for bloco de DADOS, voltar a (a), caso contrário, deverá ser FIM (5)

6. Se o bloco encontrado for um bloco de FIM, o ligador-relocador deverá somar CR a (L_1 , L_2) e montar um PLA [(L_1 , L_2)+CR] na posição conveniente de memória para que o processamento se desvie para o programa no momento da execução.

(*)Todos os erros citados são irrecuperáveis, salvo referência em contrário.

Possivelmente o PLA será na posição 10 ($0A_{16}$) de memória e será gerado um bloco do código de saída contendo apenas este PLA.

Como a tabela de subrotinas será residente em memória ela também deverá ser gerada em fita absoluta como saída do ligador-relocador. (obs.: compilado o programa principal, o PUG (subrotina) ainda está indeterminado. ~~Na fita, é necessário prever um espaço para este PUG, o qual será montado quando a subrotina em questão for alocada na memória~~)(a tabela completa deve ser gerada só no fim, quando nenhuma referência externa estiver mais indeterminada).

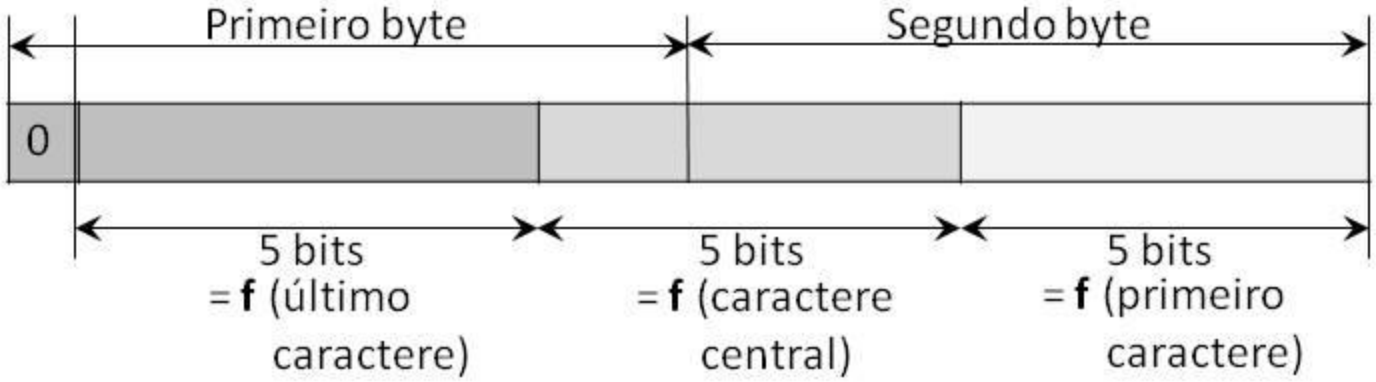
Outra coisa que o ligador-relocador deve fazer ao detectar um bloco de FIM: atualizar a constante de relocação (CR←CI) e varrer a tabela de subrotinas para verificar se há ainda alguma subrotina indefinida.

Se ainda houver subrotinas indefinidas (o número de EXTs permanece menor que o número de ENTs correspondentes), voltar a ler nova fita (6), caso contrário, terminou o processo.

6. Ler um bloco. Testar se é NOME.
   Se for => testar se é subrotina
   Se não for subrotina =>mensagem de erro (não pode haver mais de um programa principal)
   Se for subrotina, o próximo bloco deve ser ENT.
   Guardar o nome na tabela de nomes e o comprimento.
   Fazer tudo como no item 1.
   Se deu tudo certo, ler o próximo bloco (7).

7. Ler bloco.
   Se não for ENT => erro (subrotina sem entry point)
   Se for ENT => ok => Procurar na tabela de subrotinas se este ENT particular foi solicitado e está indefinido.
   Se ele estiver indefinido na tabela, o ligador-relocador deve montar o PUG (subrotina) do item 3, e na primeira palavra do NOME da SUBROTINA ligar o bit de "definido".

Obs.: a rotina de empacotamento deixa o nome da subrotina com o seguinte formato:



O bit mais significativo da primeiro byte é sempre zero, e indica símbolo indefinido. Para defini-lo basta transformá-lo em "1".

Caso o símbolo já esteja definido, é só ler o restante da fita, testando da mesma maneira os ENTs (*).

(Obs.: se nenhum dos ENTs da subrotina que está sendo carregada tiver sido chamado, o ligador-relocador deverá subtrair novamente o comprimento dos dados do CR1 e não deverá emitir código absoluto.
Se algum ENT for chamado, o CR1 deve ser mantido como estiver, e os dados deverão ser relocados, e o código absoluto correspondente deve ser gerado.

(*) Se algum bloco diferente de ENT for encontrado, vale a observação anterior.
Se houver geração de código porque houve uma chamada de ENT, deve-se testar o próximo bloco para verificar se é EXT.
Se for EXT, deve ser executado um procedimento idêntico ao descrito no item 3. (apenas se houver chamada de algum ENT da subrotina).
Terminados os EXTs, os blocos de dados devem ser tratados como em 4.
Um bloco de FIM deve apenas gerar a correção da constante de relocação (CR1→CR) e varrer a tabela de subrotinas para verificar se ainda há alguma indefinida.
Se houver alguma subrotina indefinida, voltar para 6.
Caso contrário, terminou o processo. Deve desenrolar a fita da biblioteca, e parar só ao final da fita (20 bytes nulos seguidos).

ORGANIZAÇÃO DA MEMÓRIA E ESPECIFICAÇÕES GERAIS

Ligador-relocador – Organização da Memória

endereços	conteúdo
000 a 009	Reservadas para uso pelo carregador absoluto
00A a 00B	PLA programa
00C a 00F	Outros dois ponteiros
010 a 0FF	Buffers de I/O, e áreas reservadas para transferências de parâmetros
100 a 1FF	Tabela de nomes de subrotinas, tag (in)definido e número da referência EXT
200 a ??? ??? a DFF	Área do ligador-relocador (programa, subrotinas e buffers). Fronteira variável, protegida por software. A partir da fronteira, tabela de subrotinas
E00 a F7F	Programas de utilidade geral
F80 a FFF	Loader absoluto - Fronteira protegida por hardware.

O que o Ligador-relocador deve fazer

- A ordem de entrada dos programas deve ser a seguinte: 1. Programa principal; 2. Subrotinas do usuário; 3. Subrotinas do sistema (*library*).
- O ligador-relocador deve, para economizar memória, ler bloco a bloco dos programas em formato relocável, gerar novos itens da tabela de subrotinas à medida que forem surgindo novos EXTs, atualizar as definições dos símbolos à medida que forem surgindo novos ENTs, e, durante todo esse processo, relocar o programa em questão, gerando em paralelo os *links* necessários e perfurando em fita o programa absoluto resultante em formato carregável pelo *loader* absoluto.
- Deve também fornecer códigos de erro e listagens de alocação à medida que uma nova subrotina ou programa viole qualquer das regras do jogo, ou termine a fita correspondente, respectivamente.