Uma introdução ao GAP

O nome GAP, advém de "Groups, Algorithms and Programming". Trata-se de um sistema computacional inicialmente concebido para trabalhar na área da álgebra.

O GAP é um *software* gratuito e aberto (é possível aceder aos algoritmos do sistema). É, também, extensível sendo possível ao utilizador escrever os próprios progamas e utilizá-los da mesma forma que utiliza os que fazem parte do sistema.

O GAP desenvolve-se graças à cooperação de muitas pessoas a nível internacional. Surgiu em 1986 em Aachen, na Alemanha. Em 1997, o centro de coordenação foi transferido para St. Andrews, na Escócia. Actualmente os Centros do GAP em Aachen, Braunschweig, Fort Collins e St. Andrews coordenam em conjunto o seu desenvolvimento.

É possível obter o GAP e toda a informação sobre assuntos relacionados com este sistema em `http://www.gap-system.org`.

A instalação em *Linux* é mais fácil usando a página de Frank Luebeck `http://www.math.rwth-aachen.de:8001/RsyncLinuxGAP/`

O *menuGAP* pode facilitar a utilização no *Windows*. Obtém-se em

`http://home.att.net/~menugap`

O GAP está acessível aos alunos em diversas salas:

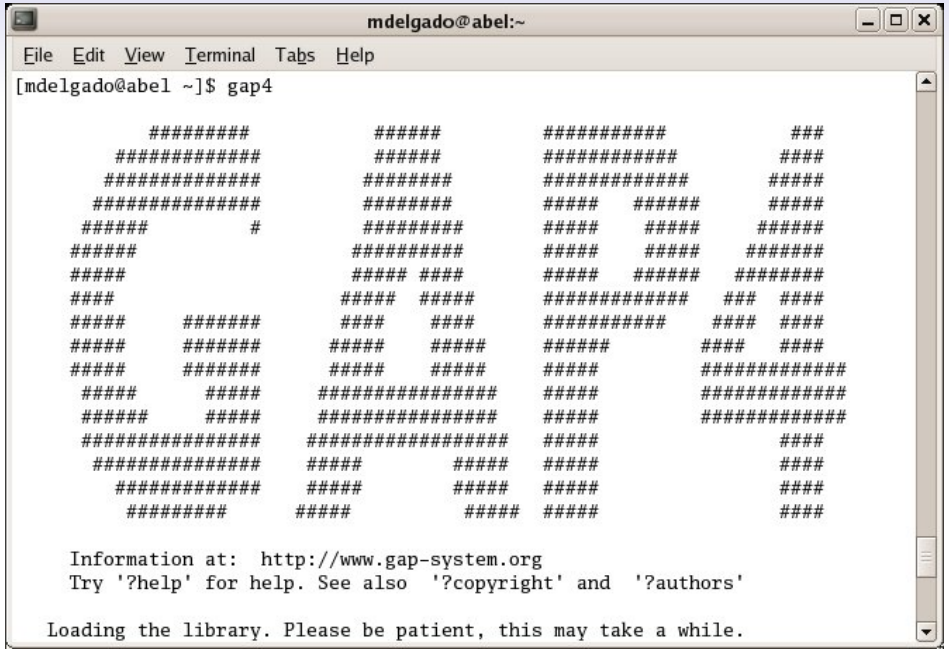
- Física: fis113 e fis021 (metade do horário está ocupado com aulas)
- Matemática(s): mp034 e mp123

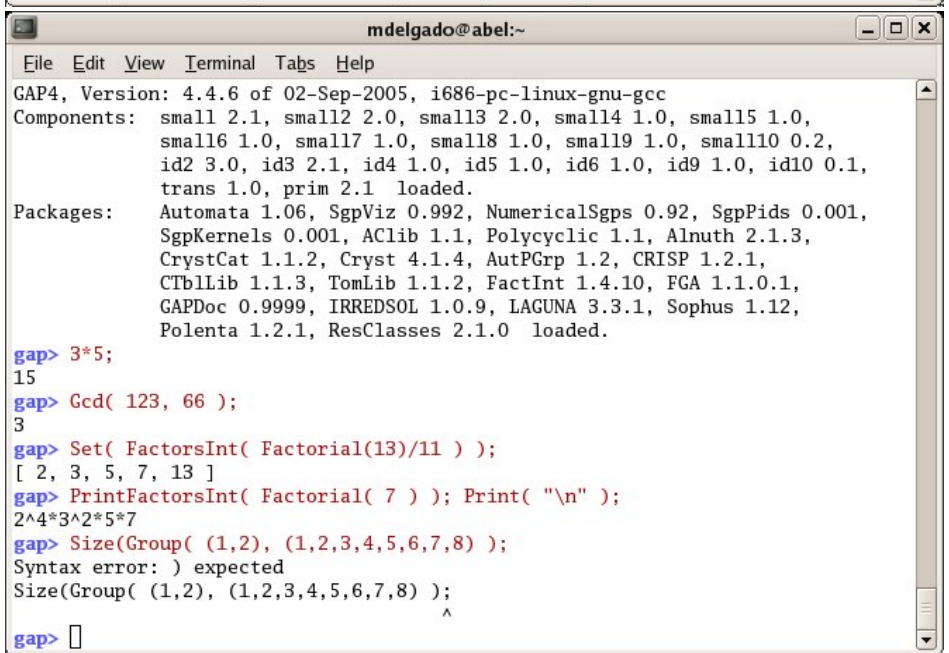
O aspecto de uma janela onde corre o GAP:

Na área de trabalho da janela do GAP, a última linha que se visualiza

gap>

está pronta a receber instruções.





Manual

O GAP tem disponível um manual de ajuda composto por 5 livros:

- *Tutorial*,
- *Reference Manual*,
- *Programming Tutorial*,
- *Programming Reference Manual* e
- *New Features for Developers*

Encontram-se subpastas dentro da pasta *doc* em versão pdf.

Dentro da pasta *doc* encontra-se ainda uma subpasta com o nome *htm*, o caminho, nas instalações usuais em ambiente Linux é `file:///usr/local/lib/gap4r4/doc/htm/index.htm` (em windows é `C:\gap4r4\doc\htm`), que disponibiliza o acesso ao conteúdo dos manuais de ajuda em versão *html*, sendo estas as versões mais cómodas e eficientes para consulta rápida e eficaz.

A linguagem GAP é considerada uma linguagem de muito alto nível, pois aproxima-se da linguagem que falamos.

Instruções básicas

Para terminar uma sessão GAP basta escrever `quit;` seguido de *return*, ou pressionar em simultâneo as teclas *ctrl-d*.
Devido a erros, o GAP entra num *break loop* ("ciclo vicioso"). Isto é indicado por

```
brk>
```

A saída do ciclo faz-se como a do próprio GAP: escreve-se `quit;` seguido de *return*, ou pressionam-se em simultâneo as teclas *ctrl-d*.

Para fazer um comentário no GAP utiliza-se o símbolo # no início da linha. Faremos também uso deste símbolo para esclarecer alguns pormenores nos exemplos dados.

Cada instrução dada deve terminar sempre com ; (ponto e vírgula) para que o GAP execute a instrução e dê a respectiva resposta. Dois pontos e vírgula ;; a seguir a uma instrução fazem com que o GAP execute a instrução mas não mostre a resposta ao utilizador.

```
gap> gap> 5+34*2; #O GAP respeita a prioridade das operacoes
73
gap> (3+9)*4;
48
gap> 5(8+9); #Esta instrucao da erro e o GAP assinala-o com ^
Syntax error: ; expected
5(8+9);
 ^
gap> 5*(8+9);
85
gap> (9-7)*(46+4;
Syntax error: ) expected
(9-7)*(46+4;
           ^
gap>
```

É possível voltar à instrução prévia recorrendo à seta "para cima". Isto permite corrigir erros sem escrever novamente as expressões.

```
gap> (9-7)*(46+4);
100
```

No GAP o nível de prioridade dos operadores aritméticos é o usual.
No primeiro nível de prioridade encontra-se o operador ^. No segundo nível estão os operadores unários + e -. No terceiro nível de prioridade estão os operadores binários *, / e mod. No último nível estão os operadores binários + e -. Deve referir-se, ainda, que os operadores aritméticos têm prioridade, quer em relação aos operadores de comparação, quer em relação aos operadores lógicos.

```
gap> 24*2-5^2;
23
gap> 45=34+11 and 45=45;
true
gap> 34<>67;
true
gap> 56=56+1;
false
```

Para guardar o trabalho a desenvolver numa sessão GAP escreve-se a instrução `LogTo(“`*caminho/nomedoficheiro*`”);`. Para parar de guardar o trabalho no ficheiro em causa escreve-se `LogTo( );`. Este ficheiro pode ser lido e editado com um editor de texto.

```
gap> LogTo("exemplo1");
gap> a:=12;
12
gap> a:=a+3;
15
gap> LogTo();
```

O comando `InputLogTo` funciona tal como o `LogTo`, só que em vez de gravar os *inputs* e *outputs*, grava apenas os *inputs*.

Normalmente torna-se mais prático elaborar os programas num editor de texto qualquer (onde se podem guardar, ler mais tarde e alterar facilmente) e copiá-los para o GAP ou lê-los com o comando `Read` no GAP. Para ler um ficheiro é necessário indicar o caminho.

```
gap> Read("algebra/exemplos/exemplo2.g");
```

Funções GAP

Um programa escrito na linguagem do GAP chama-se uma **função**. No GAP existem várias funções pré-definidas.

```
gap> Factorial(23); #Determina o factorial de 23
25852016738884976640000
gap> Gcd(18,32,8); #Maximo divisor comum
2
gap> Print("Eu gosto de Algebra","\n"); #Escreve a frase no ecra
Eu gosto de Algebra
gap> Error("Introduza numeros inteiros positivos","\n");
Error, Introduza numeros inteiros positivos
...
brk>
```

As funções `Print` e `Error` não causam automaticamente uma mudança de linha. Para começar uma nova linha depois de "imprimir" acrescenta-se "`\n`". A função `Error` utiliza-se para assinalar erros - primeiro imprime a mensagem escrita pelo utilizador, tal e qual como a função `Print`, e depois causa um *break loop*.

Podem definir-se funções de duas formas. Uma delas consegue-se utilizando os símbolos ->, como se exemplifica a seguir.

```
gap> triplica:=x->x*3;
function( x ) ... end
```

Depois de definir a função, esta pode ser aplicada a casos concretos várias vezes. No caso do exemplo, vamos calcular o triplo de 15.

```
gap> triplica(15);
45
```

A outra forma de definir funções tem a seguinte sintaxe:

nomedafunção`:=function(`*argumentos*`)`
instruções
`end;`

Está exemplificada a seguir.

```
gap> nome:=function(n)
> Print("O meu nome e ",n,"\n");
> end;
function( n ) ... end
gap> nome("Joana"); #Ja estamos a utilizar a funcao
O meu nome e Joana
gap> nome("Dinis");
O meu nome e Dinis
```

Os nomes das funções que fazem parte do GAP começam sempre por maiúsculas. Assim, funções cujos nomes comecem por uma letra minúscula não terão problemas de compatibilidade.

Para completar a edição das funções, quer do GAP, quer definidas pelo utilizador, pode usar-se a tecla *tab* depois de digitar as primeiras letras do nome da função pretendida.

No GAP como noutras linguagens de programação, existem as seguintes frases pré-definidas: *if* (se ... então); *while* (enquanto ... faz); *repeat* (repete ... até); *for* (para cada objecto da lista faz ...). A sintaxe da frase *if* é:

- `if` *condição* `then` *instruções* `fi;`

A parte *else* pode ser omitida.
É possível escrever uma frase *if* dentro de outra várias vezes, tanto de forma completa como de forma abreviada.

- `if` *condição* `then` *instruções* `else if` *condição* `then` *instruções* `else` *instruções* `fi; fi;`
- `if` *condição* `then` *instruções* `elif` *condição* `then` *instruções* `else` *instruções* `fi;`

As frases *while*, *repeat* e *for* também são conhecidas por ciclos por executarem ciclicamente o mesmo conjunto de instruções. A sintaxe:

- `while` *condição* `do` *instruções* `od;`
- `repeat` *instruções* `until` *condição*`;`
- `for` *variável* `in` *lista* `do` *instruções* `od;`

Exemplo

Quando a é um número inteiro e b é um inteiros positivo, `a mod b` é o resto da divisão inteira de a por b.

```
gap> parouimpar:=function(n) #Indica a paridade de n
>       local p; #Introducao da variavel local p
>       if n mod 2 = 0 then
>          p:="Par";
>       else
>          p:="Impar";
>       fi;
>       return p;
>    end;
function( n ) ... end
gap> parouimpar(23); #Inicio da utilizacao da funcao
"Impar"
gap> parouimpar(54);
"Par"
gap> parouimpar(0);
"Par"
```

A seguir vamos utilizar a função `parouimpar` criada antes. O GAP só utiliza esta função se ela for executada previamente. Salienta-se, ainda, o facto de no exemplo anterior o resultado ser devolvido com a instrução `return`, o que permite que possa ser utilizado mais tarde.

Exemplo

A função `soma1` determina a soma dos números pares entre 1 e *m* inclusive. Utilizou-se um ciclo *while*.

```
gap> soma1:=function(m)
> local i,s;
> i:=1; s:=0;
> while i<=m do
>    if (parouimpar(i) = "Par")  then
>        s:= s+i;
>    fi;
>    i:=i+1;
> od;
> return s;
> end;
function( m ) ... end
gap> soma1(1000);
250500
gap>
```

Exemplo

A função soma2 determina a soma dos números pares entre 1 e *m* inclusive. Neste exemplo utilizou-se um ciclo *repeat* para permitir uma comparação com o ciclo *while* presente no exemplo anterior.

```
gap> soma2:=function(m)
> local i,s;
> i:=1; s:=0;
>     repeat
>         if parouimpar(i) = "Par" then
>         s:= s+i;
>         fi;
>     i:=i+1;
>     until i>m;
> return s;
> end;
function( m ) ... end
gap> soma2(170000);
7225085000
gap>
```