GAP – bardzo krótkie wprowadzenie

Rafał Lutowski

1. Uwagi wstępne

Instalacja GAPa w systemie Windows

1. Strona domowa GAPa: `www.gap-system.org`.
2. Z podstrony „Downloads” (`www.gap-system.org/Releases/index.html`) wybieramy plik wersji instalacyjnej dla Windowsa. Aktualnie jest to plik

   ```
   gap-4.10.1-p1.exe
   ```

3. Jako folder instalacyjny wskazujemy

   ```
   D:\gap4r7
   ```

   Odznaczamy instalację niewymaganych pakietów.

   Uwaga: Nazwa folderu.
4. Omówienie struktury folderów. Pliki `gap.bat` i `gap.sh`.

Kilka uwag na temat składni

1. Wielkość liter ma znaczenie.
2. Komendy kończymy znakiem ’;’.
3. Jeżeli chcemy „złamać” linię komendy, używamy znaku ’\’.
4. Pomoc na temat funkcji uzyskujemy wpisując jej nazwę, poprzedzoną znakiem zapytania, np.

   ```
   gap> ?LogTo
   ```

   Może być kilka opcji do wyboru. Wtedy używamy np.

   ```
   gap> ?1
   ```

   Wychodzimy wciskając ’q’. Menu pomocy na dole.

Wczytywanie komend z plików

1. Polecenie

```
gap> Print("Hello World!\n");
```

2. Wpiszmy to polecenie do pliku

```
D:\gap\cwiczenie01.g
```

3. Polecenie

```
gap> Read("D:\\gap\\cwiczenie01.g");
```

Readline i historia

1. Readline: używanie klawisza Tab do dopełniania poleceń.

   Ćwiczenie: Wpisz `Prin` i naciśnij Tab dwa razy, z przerwą.

   Ćwiczenie: Wpisz `Print` i naciśnij tab dwa razy.

2. Historia: użycie klawiszy `Up` i `Down` pozwala na poruszanie się w historii poleceń.

Zapisywanie pracy

1. Funkcja

```
SaveWorkspace
```

   zapisuje aktualną sesję GAPa.

2. Ponieważ powyższa funkcja nie zapisuje historii, możemy to wykonać ręcznie, przy pomocy funkcji

```
SaveCommandLineHistory
```

   i wczytać przy pomocy funkcji

```
ReadCommandLineHistory
```

Błędy

1. Po próbie dzielenia przez 0

```
gap> 1/0;
```

oprócz wyświetlenia błędu, pojawi się nowy „tekst zachęty”

```
brk>
```

Oznacza to, że znajdujemy się „w środku” przerwanego polecenia. Wychodzimy z niego, wpisując

```
brk> quit;
```

2. „Pętle przerwania” (break loops) mogą się zagnieżdżać.

2. Język programowania

1. Znaki specjalne

```
"    ‘    (    )    *    +    ,    -    #
.    /    :    ;    <    =    >    ~
[    \    ]    ^    _    {    }    !
```

2. Operatory

```
+    -    *    /    ^    ~    !.
=    <>   <    <=   >    >=   ![
:=   .    ..   ->   ,    ;    !{
[    ]    {    }    (    )    :
```

Przypisanie

```
:=
```

Porównanie

```
= <> > < >= <=
```

Arytmetyka

```
+ - * / mod ^
```

3. Słowa kluczowe

```
gap> GAPInfo.Keywords;
```

4. Identyfikatory: litery, cyfry, '_', '@'; przynajmniej jedna litera. Można użyć '\', aby użyć innych znaków.

5. Zmienne globalne, lokalne i funkcje. W pliku

   ```
   D:\gap\cwiczenie02.g
   ```

   umieśćmy następujący ciąg poleceń GAPa:

   ```
   g := 0; # zmienna globalna g
   x := function ( a, b, c )
     local y;
     g := c; #
     y := function ( y )
       local d, e, f;
       d := y; #
       e := b; #
       f := g; #
       return d + e + f;
     end;
     return y( a ); #
   end;
   ```

   Następnie wykonajmy polecenia:

   ```
   gap> x(1,2,3);;
   gap> x(1,2,3);
   ```

6. Jeżeli:

   ```
   if bool-expr1 then
     statements1
   {                    # \
   elif bool-expr2 then # kilka
     statements2        # opcjonalnych
   }                    # /
   [                    # \
   else                 # jedno
     statements3        # opcjonalne
   ]                    # /
   fi;
   ```

Wersja jedno-liniowa:

```
if bool-expr1 then statements1 { elif bool-expr2 then statements2 }[ else
statements3 ] fi;
```

Ćwiczenie: Napisz funkcję znak, która zwraca $-1, 0, 1$ w zależności od tego, jaki znak ma jej argument.

7. While:

```
while bool-expr do statements od;
```

Repeat:

```
repeat statements until bool-expr;
```

Ćwiczenie: Silnia, wykorzystując obie konstrukcje.

8. For:

```
for simple-var in list-expr do statements od;
```

W szczególności, „standardowa” pętla od do ma postać

```
for variable in [from..to] do statements od;
```

Ćwiczenie: Silnia z użyciem for.

Iteratory:

```
for variable in iterator do statements od;
```

Obiekty, które możemy enumerować:

```
for variable in object do statements od;
```

Ćwiczenie: Wykorzystując funkcje `SignPerm` oraz `SymmetricGroup` wypisz znaki wszystkich permutacji grupy S_3.

9. `break` i `continue`

Ćwiczenie: Wykorzystując funkcję `Order` oraz `break` wypisz pierwszy napotkany przez pętlę for element rzędu 6 grupy S_6.

Ćwiczenie: Wykorzystując `continue` wypisz wszystkie elementy S_3 rzędu różnego od 3.

10. Używając w definicji funkcji specjalnego argumentu `arg` (jest to lista) konstruujemy funkcję z nieokreśloną liczbą argumentów.

    Ćwiczenie: Napisz funkcję liczącą sumę jej argumentów.

    Uwaga: Następujące dwie konstrukcje definiują taką samą operację:

```
kw1 := function(x) return x^2; end;
kw2 := x -> x^2;
```

Listy

1. Przykłady list:

```
a := [1..10];
b := [[1],2,3];
c := [ [1,2,3], [4,5,6] ];
d := [ [1,2,3], [4,5] ];
e := [ (), 1, false ];
f := [1,,,,1];
```

   Ćwiczenie: Na podanych listach wykonaj polecenia `IsList`, `IsDenseList`, `IsHomogenousList`, `IsTable`, `IsMatrix`.

2. Dostęp do elementu/ów list:

```
gap> a[1];
gap> b[1];
gap> b[1][1];
gap> b{[2,3]};
```

   Ćwiczenie: Wyznacz macierz powstałą z macierzy `c` po usunięciu drugiej kolumny.

3. Funkcja `List`:

```
gap> List([1..10], function(k) return k^2; end);
gap> List(SymmetricGroup(3), SignPerm);
```

   Ćwiczenie: Używając listy `Primes`, wyświetl kwadraty pierwszych 15 liczb pierwszych. Użyj bardziej kompaktowej definicji funkcji (operator `->`).

   Ćwiczenie: Utwórz listę elementów postaci $[\sigma, \mathrm{sgn}(\sigma)]$, gdzie $\sigma \in S_4$.

4. Modyfikowanie list - funkcje `Add`, `Remove`, `Append`.

5. Sprawdzanie przynależności:

```
gap> 1 in c;
gap> 1 in c[1];
```

6. Wybrane funkcje dla list:

```
Concatenation, Compacted, Collected, Flat, Reversed, Maximum, Minimum
Cartesian, List, Length, Size, Filtered, First, ForAll, ForAny, Sum
```

Ćwiczenie: Używając funkcji `Collected`, wyświetl statystyki reszt z dzielenia przez 3 wśród liczb z listy `Primes`.

Ćwiczenie: Używając funkcji `Filtered`, wyświetl liczby Mersenne'a z listy `Primes`. W sprawdzaniu ogranicz się do zbioru $M = \{2^k - 1 : 1 \leqslant k \leqslant 20\}$.

Ćwiczenie: Dla zbioru M z poprzedniego ćwiczenia wyznacz wszystkie liczby Mersenne'a.

Ćwiczenie: Wypisz wszystkie elementy grupy S_5 rzędu 6.

Ćwiczenie: Wyznacz największy rząd elementu grupy S_7.

Ćwiczenie: Czy w grupie A_8 istnieje element rzędu 12?

Ćwiczenie: Używając funkcji `AsSet` oraz `Set`, wyświetl wszystkie możliwe rzędy elementów grupy A_8.

7. Przykłady definiowania zakresów:

```
gap> [1..10];
gap> [1,2..10];
gap> [1,3..9];
```

Uwaga na:

```
gap> [1,3..10];
```

3. Grupy. Wprowadzenie

Permutacje

1. Permutacje są zapisywane jako iloczyn rozłącznych cykli. Składanie odbywa się poprzez mnożenie permutacji.

```
gap> a := (1,2,3)(4,5,6);
gap> b := (1,2,3)(2,3,4);  # zle - cykle rozlaczne
gap> b := (1,2,3)*(2,3,4); # skladanie - ok
gap> One(a);               # element neutralny
```

```
gap> Inverse(a) = a^-1;    # element odwrotny
```

2. Działanie permutacji:

```
gap> OnPoints(1, (1,2));
gap> 1^(1,2);
```

Ćwiczenie: Napisz funkcję liczącą wyznacznik macierzy metodą permutacyjną.

3. Liczba punktów zmienionych przez permutację `a`:

```
gap> NrMovedPoints(a);
```

Ćwiczenie: Przypomnienie: Charakterem reprezentacji permitacyjnej stowarzyszonej z „naturalnym" działaniem S_n na zbiór $\{1, \ldots, n\}$ jest funkcja $f\colon S_n \to \mathbb{C}$, przypisująca permutacji liczbę jej punktów stałych. Liczba podreprezentacji trywialnych dana jest wzorem

$$\frac{1}{|S_n|} \sum_{\sigma \in S_n} f(\sigma).$$

Pokaż dla $n = 5, 6, 7$, że reprezentacja ta zawiera podreprezentację trywialną oraz że nie jest ona nieprzywiedlna.

Grupy

1. Tworzenie:

```
gap> G := Group((1,2,3)(4,5), (1,2));
```

2. Generatory grupy:

```
gap> gens := GeneratorsOfGroup(G);
```

Uwaga: Nie zawsze generatory przekazane funkcji `Group` będą faktycznie zapisane jako generatory i zwracane przez powyższą funkcję. Aby wymusić zapisanie „naszych" generatorów używamy funkcji `GroupWithGenerators`.

3. Jeżeli wyznaczone są generatory, to możemy się do nich odwołać bezpośrednio:

```
gap> G.1;
```

4. Podgrupy, rząd grupy:

```
gap> G := Group( (1,2,3)(4,5), (1,2) );
gap> Size(G);
gap> H := Subgroup(G, [(2,3),(4,5)]);
gap> Parent(H) = G;
gap> H := Group( (2,3), (4,5) );
gap> IsSubgroup(G, H);
gap> Parent(H) = G;
gap> Index(G,H);
gap> H := AsSubgroup(G, H);
gap> Parent(H) = G;
```

5. Przykłady funkcji dla podgrup:

```
IsNormal, IsCharacteristicSubgroup, IsSubnormal
```

6. Wyświetlanie struktury grupy:

```
gap> StructureDescription(G);
gap> M := MathieuGroup(23);
gap> StructureDescription(M);
```

Uwaga: Często wynik zwrócony przez funkcję jest niejednoznaczny – należy go traktować z należytą ostrożnością:

```
gap> grp1 := SmallGroup(32,11);
gap> grp2 := SmallGroup(32,24);
```

Ćwiczenie: Sprawdź, że powyższe grupy mają taki sam opis w funkcji `StructureDescription`, ale nie są to grupy izomorficzne (np. centrum).

7. Warstwy:

```
gap> r := RightCoset(H, (1,2));
gap> Representative(r);
gap> (1,2,3) in r;
gap> (2,3) in r;
gap> List(r);
```

Wszystkie warstwy podgrupy i ich reprezentanty:

```
gap> RightCosets(G, H);
gap> RightTransversal(G, H);
```

8. Enumerowanie podgrup:

```
gap> AllSubgroups( G );
```

   Dla dużych grup lepiej poszukać klas sprzężoności podgrup

```
gap> ConjugacyClassesSubgroups(G);
```

9. Klasy sprzężoności elementów grupy

```
gap> NrConjugacyClasses(G);
gap> cc := ConjugacyClasses(G);
gap> Order( Representative(cc[2]) );
```

   Ćwiczenie: Wyznacz wszystkie możliwe rzędy elementów grupy S_{20}.

10. Wybrane funkcje testujące własności grupy:

```
IsCyclic, IsElementaryAbelian, IsAbelian, IsNilpotentGroup,
IsSolvableGroup, IsPolycyclicGroup, IsPerfectGroup, IsSimpleGroup
```

11. Komutator i abelianizacja

```
gap> D := DerivedSubgroup(G);
gap> IsNormal(G, D);
gap> StructureDescription( G/D );
gap> CommutatorFactorGroup = G/D;
gap> AbelianInvariants(G);
```

   Ćwiczenie: Sprawdź, że grupy `grp1` i `grp2` mają różną abelianizację oraz eksponenty.

12. Biblioteka grup skończonych:

```
gap> grp1 := SmallGroup(32, 11); # to bylo wyzej
gap> IdSmallGroup(grp1);         # rowniez IdGroup
gap> IsGroup(SymmetricGroup(3));
gap> IdGroup(CyclicGroup(6));
gap> AllSmallGroups(6);
```

Homomorfizmy grup

1. Wyznaczanie homomorfizmu poprzez obrazy elementów (domyślnie generatorów):

```
gap> A1 := Group( (1,2)(3,4), (1,3)(2,4) );
gap> A2 := AbelianGroup([2,2]);
gap> hom := GroupHomomorphismByImages(A1, A2);
gap> IsInjective(hom);
gap> IsSurjective(hom);
gap> Image(hom, (1,4)(2,3));
```

Ćwiczenie: Napisz funkcję `IsGroupIsomorphism`, która będzie sprawdzała, czy dana funkcja jest izomorfizmem grup.

Ćwiczenie: Niech `S1` będzie grupą generowaną przez permutacje $(1,2),(2,3)$, a `S2` – grupą o identyfikatorze `[6,1]`. Skonstruuj izomorfizm tych grup.

2. Wyznaczanie homomorfizmu poprzez funkcję

```
gap> hom := GroupHomomorphismByFunction(G, A2,
> function(x)
> if SignPerm(x)=-1 then return A2.1; else return One(A2); fi;
> end );
gap> Index( Range(hom), Image(hom) );
gap> Source(hom) = G;
```

Ćwiczenie: Wykorzystując powyższe funkcje (obie) skonstruuj homomorfizm „znak permutacji" z grupy G do grupy $\mathrm{GL}(1,\mathbb{Z})$ (w GAPie `GL(1,Integers)`). Wyznacz jądro tego homomorfizmu.

3. Przeciwobrazy. Załóżmy, że w powyższym ćwiczeniu nazwa homomorfizmu to `sgn`.

```
gap> PreImage(hom, GL(1,Integers));
gap> PreImagesRepresentative(sgn, [[-1]]);
```

Uwaga: Działająca wersja powyższych komend – użycie `Group([[-1]])`.

```
gap> gl := Group([[-1]]);
gap> sgn := ...
gap> Kernel(sgn) = PreImage(sgn, TrivialSubgroup(gl));
```

4. Abelianizacja:

```
gap> h1 := MaximalAbelianQuotient(G);
gap> D := DerivedSubgroup(G);
gap> h2 := NaturalHomomorphismByNormalSubgroup(G, D);
```

Ćwiczenie: Sprawdź, że w istocie `h1` i `h2` są sobie równe.