Pointeurs et structures de données

Les pointeurs en Perl:

- Pointeur sur scalaire

```
$p_var = \$ma_variable;
```

- Pointeur sur tableau :

```
$p_tab = \@mon_tableau;
```

- Pointeur sur tableau associatif :

```
$p_hash = \%mon_tabl_associatif;
```

- Pointeur sur fonction :

```
$p_fun = \&ma_fonction;
```

Références anonymes

- Pointeur sur tableau anonyme:

```
$p_tab = [1,2,3,4,5];
```

- Pointeur sur hachage anonyme:

```
$p_hash = {Pierre => 12,Paul => 17};
```

- Pointeur sur fonction anonyme:

```
$p_fun = sub {print shift};
```

- Utilisation:

```
$p_var-> = 1;
$y= p_tab->[3];
print $p_hash->{Pierre};
$p_fun->("Bonjour\n");
```

Association (clé, liste de valeurs...)

- Principe : on associe une liste de valeurs à une clé
- Exemple : les valeurs du tableau associatif sont des pointeurs sur tableau anonyme :

```
$instrument{John} = ["Guitare", "Piano"];
$instrument{Paul} = ["Basse", "Guitare", "Piano"];
$instrument{George} = ["Guitare"];
$instrument{Ringo} = ["Batterie"];
```

- Utilisation :

```
print $instrument{Paul}->[1];   # affiche Guitare
```

Association (Clé,Attributs)

- Dans une base de données, il est courant de définir un schéma de données (clé, attributs)
- Exemples:
  - Client(**num_client**,nom, prénom, adresse)
  - Ouvrage(**référence**, Titre, Auteurs)
- En Perl :

```
Client{1}=["Dupont","Pierre","2, rue des lilas"];
Client{2}=["Durand","Paul","7, rue des vergers"];
Client{3}=["Dubois","Jacques","15, rue des postes"];
etc...
```

Objets composés

- Enregistrement = Variable composée
- Exemple1: objet nombre complexe

```
$z={}; % pointeur sur tab associatif anonyme
$z->{reel}=2.0;
$z->{imag}=1.0:
```

- Exemple 2: objet eleve

```
$e={}; % pointeur sur tab associatif anonyme
$e->{nom} = "Dupont";
$e->{prenom} = "Pierre";
$e->{age} = 22;
$e->{moyenne} = 12.5;
```

Arbre binaire

```
sub inserer { my($a,$val)=@_;
   if ($a == undef) {
                $a = {};
                $a->{gauche}=undef;
                $a->{droit}=undef;
                $a->{val}=$val;
                } else {
        if ($val < $a->{val}) {
                $a->{gauche}=inserer($a->{gauche},$val);
        } else {
                $a->{droit}=inserer($a->{droit},$val);
        }
   }
   return $a;
}
```

Arbre binaire suite

```
sub afficher { my ($a) = @_;
   if ($a!=undef) {
        afficher ($a->{gauche});
        print "$a->{val}\n";
        afficher ($a->{droit});
   }
}

% Programme principal

my $a=undef;

foreach (1..20) {
   $a=inserer($a,int(rand(1000)));
}

afficher($a);
```

Les modules en Perl

Présentation

- Un module est un ensemble de variables et de fonctions rassemblées dans un fichier
  - dont l'objectif est d'enrichir la bibliothèque standard du langage.
  - Chaque module doit être impérativement terminé par une ligne contenant 1.
- Des centaines de modules ont été créés, certains très spécialisés, qui sont mis à la disposition sur le site du CPAN.
- Un module `MonModule` est enregistré dans un fichier à son nom `MonModule.pm`, suivi de l'extension `.pm`. Pour être utilisé dans un programme, il doit être inclus dans le code du programme par la directive :

```
use MonModule;
```

Module

- Un module
  - est un regroupement thématique de variables et de fonctions
  - On parle aussi de librairie de fonctions
- Programmation modulaire :
  - Faciliter la réutilisation des fonctions
  - Documentation
  - Partage de « briques » facilitant la réalisation d'applications plus évoluées
  - Approche incrémentale (ne pas refaire ce qui a déjà été fait)

Classe/Module/Paquetage

```
Package Arbre;
```

- Tout symbole déclaré à la suite d'une déclaration de paquetage est implicitement préfixé par le nom du module `Arbre::`

```
Package Arbre;
$a = &inserer (12);
```

Le nom réel est `$Arbre::a` et la fonction appelée est `&Arbre::inserer`.

Installation

- Pour tester l'installation et la disponibilité d'un module, le plus direct est de lancer la directive d'inclusion en ligne de commandes :
  - Exemple : le module `Math::BigInt`, pour manipuler des entiers longs, est-il disponible ?

```
$ perl –e 'use Math::BigInt';
```

- Si le fichier n'est pas placé dans l'un des chemins contenus dans la liste système `@INC`, on peut l'y ajouter par une directive :

```
use lib : use lib '/chemin/vers/rep_module';
use MonModule;
```

Documentation

- Consulter la doc d'un module avec `perldoc`
  En principe chaque module a été installé avec sa documentation spécifique.
- L'utilitaire **perldoc** permet de l'ouvrir et de la parcourir.
- Exemples :

```
$ perldoc DBI
$ perldoc DBD::mysql
<q pour sortir>
```

Installation manuelle et test sous Unix

- doc à `http://www.perl.com/CPAN/modules/INSTALL.html`
- téléchargement d'un module à
  `http://www.perl.com/CPAN/modules/by-module/`
- décompresser : `gzip -d module.tar.gz`
- examiner l'archive : `tar -tf module.tar`
- désarchiver : `tar -xvf module.tar`
- aller dans le répertoire créé : `cd module-..`
- compiler, installer et tester

```
perl Makefile.PL
make
make test
make install
$ perl -e 'use module';    pour tester
```

Exemple d’utilisation

```
use IO::Socket;

$socket = IO::Socket::INET->new(     PeerAddr =>  "capucine",
                                PeerPort => 1234,
                                Proto    => "tcp",
                                Type     => SOCK_STREAM)
   or die "Impossible de se connecter : $@\n";
$socket ->autoflush(1);

$socket->send ("Hello!");

$socket->recv($reponse,300);

print "voici la reponse du serveur : $reponse \n";

close($socket);
```

L’approche objet

Définition

- La **programmation orientée objet**
  - est une méthode de programmation
  - Fondée sur l'idée d'encapsulation :
  - Un objet est une structure de donnée
    - Dont les valeurs (états) sont cachés
    - Accessible par des fonctions dédiées (appelées « méthodes »)
  - Ainsi :
    - On cache ce qui est difficile (les traitements)
    - On protège certaines données « sensibles »

Classe

- Une classe est un descripteur d'objet
- POO : les programmes sont écrits sous forme de classes.
- La clase est le « moule » à partir duquel les objets sont « forgés »
  - Un objet est une « instance de classe »
- Une classe, c'est :
  - Un nom
  - Une interface :
    - Variables publiques
    - Noms et paramètres des méthodes
  - Une implémentation :
    - la réalisation de l'objet lui-même :
    - Sous forme de variables et de fonctions

Les objets en Perl

- En Perl, les Objets sont réalisés à partir des éléments préexistants du langage :
  - Un objet est une référence (un pointeur).
    - Il s'agit en général d'un pointeur sur un tableau associatif anonyme.
  - Une classe est un paquetage
    - Les fonctions du paquetage implémentent les méthodes de la classe
  - Une méthode est une fonction
    - Déclarée dans le paquetage en question
    - Mode d'appel : `$a -> ma_methode (arguments)`

Les méthodes

- Une « méthode » est une fonction associée à un objet
- Il existe deux catégories de méthodes :
  - Méthode de classe:
    - Pas besoin d'instancier un objet
    - La fonction est « directement » utilisable
  - Méthode d'instance:
    - Associée à une instance (un objet particulier)
    - Sert à manipuler les variables propres à cet objet

Invocation de méthode

- Syntaxe

  invoquant -> methode (liste)

- Méthode de classe

```
use Arbre;
$a = Arbre -> new (12);
% equivalent de $a = new Arbre (12)
```

- Méthode d'instance

```
$a -> inserer (17);
```

Constructeur/destructeur

- Le constructeur est une méthode de classe :
  - Servant à réaliser un objet particulier (à « instancier » un objet)
  - Réservation d'une zone mémoire propre à cet objet
- Le destructeur est une méthode d'instance :
  - Qui supprime l'objet et sa zone mémoire associée.

Construction d'objet

- Principe : « consacrer » un objet à une classe :*bless*

```
$mon_arbre = {};
bless($mon_arbre,"Arbre");
```

- Constructeur de classe

```
package Arbre;
sub new { my ($classe) = @_;
        my $self = {};
        bless($self,$classe);
        return $self;
}
```

- Utilisation

```
use Arbre;
$mon_arbre = Arbre -> new ;
```

Module Arbre binaire

```
package Arbre;

sub new { my ($classe,$val)=@_;
   my $a={};
   $a->{gauche}=undef;
   $a->{droit}=undef;
   $a->{val}=$val;
   bless($a,$classe);
   return $a;
}
```

Module arbre binaire suite

```
sub inserer { my($a,$val)=@_;
   if ($a==undef) {
                $a = Arbre->new ($val);
                } else {
        if ($val < $a->{val}) {
                $a->{gauche}->inserer($val);
        } else {
                $a->{droit}->inserer($val);
        }
   }
   return $a;
}
```

Utilisation

```
use Arbre;
$a = Arbre -> new (12) ;

foreach (1..20) {
   $a->inserer(int(rand(1000)));
}

$a->afficher;
```

Perl/Tk

Une introduction aux interfaces graphiques avec PERL

Introduction

- Tk est un gros module offrant une API (Application Programming Interface) de développement d'interface graphique.
- D'abord écrit comme extension du langage Tcl, il a été adapté pour Perl (et d'autres langages dont Python ..)
- Il fonctionne sur des plate-formes graphiques aussi variées que X11, MacOS et Win32.
- C'est un langage assez simple à apprendre et permettant de créer des interfaces graphiques **événementielle** en peu de lignes.

Premier exemple

```
#!/usr/bin/perl

use Tk;

MainWindow->new->Button(-text=>"Bonjour le monde !",
                        -command=>sub {exit}) ->pack();
MainLoop;
```

Installation et test

- Installation Unix/Linux
  - récupérer la dernière version du module sur le CPAN, et le compiler
    http://www.perl.com/CPAN-local/modules/by-category/08_User_Interfa
- test

```
$ perl -e 'use Tk';
```

ne renvoie pas de message d'erreur si installé

Installation sous windows

- version Perl Active State sous Windows
  - Se connecter au site `http://www.activestate.com`, puis à la section *téléchargement pour Windows MSI*
  - Télécharger l'archive ActivePerl (environ 8 Mo) dans un répertoire C:\Perl puis procéder à l’installation.

Widget : une brique de base

- Tk permet de créer et de gérer des widgets et toutes les choses qui ressemblent à une interface graphique
- Un Widget est une brique de base manipulée dans une interface graphique
- Créer une interface graphique avec Perl/Tk c'est :
  - créer, placer, manipuler des widgets.

Différents types de widget

- On peut scinder les widgets en 2 types :
  - les conteneurs : ceux qui peuvent contenir d'autres widgets :
    - fenêtres,
    - cadres,
    - menus,
    - listes de choix…
  - les widgets de base : ceux qui ne contiennent pas d'autres widgets
    - boutons,
    - cases à cocher,
    - boutons radio,
    - barres de défilement...

Création d'un widget

- Un même principe de base :
  - chaque widget doit avoir un parent qui le surveille et en garde une trace durant son existence
- Un exemple pour commencer
  - on suppose que le widget **`$parent`** existe déjà. Pour créer un widget de type **`widgetType`** :

```
$fils = $parent -> widgetType(-option => valeur, ... );
```

Structuration du programme

- Partie 1 : de la première ligne à « MainLoop; »
  - création de la fenêtre principale et de tous les widgets (fils)
  - l'instruction **MainLoop** met en marche le gestionnaire d'événements : l'application ne fera que ce qu'on lui dit de faire via les widgets définis
- Partie 2 : le reste
  - toutes les procédures appelées par les widgets définis ;
  - et les procédures appelées par ces procédures
  - ...

Exemple détaillé

```
#!/usr/bin/perl -w
# Programme disant juste "bonjour" !

# chargement du module
use Tk;

# création de $fen, l'objet fenêtre principale, par
# invocation du contructeur sur la classe MainWindow

$fen = MainWindow -> new();

# syntaxe d'appel des méthodes (fonction s'adressant à
   un objet)
# la méthode minsize impose une taille minimum à
   l'objet fenêtre
$fen -> minsize('250','60');
```

Exemple détaillé (suite)

```
# la méthode title de l'objet $fen est chargée d'afficher le
   titre
$fen -> title("Bonjour");

# la méthode Label affiche une zone de texte non modifiable.
$texte = $fen -> Label( -text => "Bonjour tout le monde !");

# pack est un gestionnaire de disposition, qui place le
   composant dans la fenêtre
$texte -> pack;

# la méthode Button affiche un bouton, avec le label Fermer.
# on associe une commmande à l'action "clic" sur ce bouton
$bouton = $fen -> Button( -text => "Fermer",
                          -command => sub {exit});
$bouton -> pack;
# appel au gestionnaire d'événements
# il est chargé de l'écoute (en boucle) des événements et du
   déclenchement des actions
MainLoop;
```

Création de la fenêtre principale

- **`$mw= MainWindow->new;`**
  - cette ligne de code permet de créer la fenêtre principale qui contiendra les autres widgets
- MainWindow (1), TopLevel (2), Frame (3) :
  - (1) : fenêtre principale
  - (2) : premier niveau.
    - **`$top=$mw->Toplevel();`**
    - (1) est une version spéciale de (2). (1) est la première fenêtre créée dans un programme
  - (3) : la widget Frame (cadre)
    - **`$cadre=$mw->Frame(-borderwidth => 2);`**
    - un conteneur invisible qui sert principalement à ranger des widgets

Mise à jour de la fenêtre principale

```
$mw->title("Recherche avec Expressions
  régulières");
```

- A la variable `$mw` est associée un objet complexe (la fenêtre principale) dont on modifie le titre

Création d'un cadre et insertion de widgets dans ce cadre

```
$cadre = $mw->Frame->pack(-side => 'top',
                          -fill => 'x');
$cadre->Label(-text => "File : ", -relief =>
   'raised')->pack(-side => 'left',-anchor =>
   'w');
$cadre->Entry(-textvariable => \$nom_fic)->pack(-
   side => 'left', -anchor => 'w', -fill => 'x',
   -expand => 1);
$cadre->Button(-text => "load", -command => sub
   {&lire_fic})->pack(-side => 'left', -anchor =>
   'e');
```

Création et affichage du cadre

```
$cadre = $mw->Frame->pack(-side =>
   'top',-fill => 'x');
```

- Le code :

  `$cadre = $mw->Frame;`

  – permet de créer un cadre

- Affichage du cadre :
  - l'affichage d'un widget se fait avec un gestionnaire d'espace, en général **`pack`**
  - Pour afficher un widget :
    - **`$cadre->pack();`**

Paramètres d'affichage

```
$cadre = $mw->Frame->pack(-side => 'top',-fill =>
   'x');
```

- l'option **-side** spécifie le bord de la fenêtre ou du cadre sur lequel est placé le widget,
  - ici ' top '
- l'option **-fill** force le widget à remplir dans la direction spécifiée le rectangle qui lui a été alloué :
  - ici la largeur de la fenêtre principale

Création d'un widget Etiquette

```
$cadre->Label(-text => "File : ",
              -relief => 'raised')
             ->pack(-side => 'left',
                    -anchor => 'w');
```

- Le code ci-dessus associe au widget $cadre un widget Etiquette (**Label**)
  - ce widget ressemble à un bouton
  - il contient du texte, il peut être mis en relief, avoir une font différente...

Paramètres du widget Etiquette

```
$cadre->Label(-text => "File : ",
              -relief => 'raised');
```

- l'option `-text` indique le texte affiché dans le widget
- l'option `-relief` modifie le type de contour tracé autour de l'étiquette

Paramètres d'affichage

```
$cadre->Label(-text => "File : ",
              -relief => 'raised')
             ->pack(-side => 'left', -anchor
  => 'w');
```

- l'option -anchor ancre le widget dans le rectangle qui lui a été alloué

Création d'un widget de saisie

```
$cadre->Entry(-textvariable => \$nom_fic)->pack(-
   side => 'left', -fill => 'x', -expand => 1);
```

- Le widget de saisie (**Entry**) permet à l'utilisateur de taper du texte
- L'option **-textvariable** permet de savoir ce que l'utilisateur a tapé dans le widget ; le contenu est associé à la variable associée à cette option
    - ici la variable **$nom_fic**

Paramètres d'affichage

```
$cadre -> Entry(-textvariable => \$nom_fic) ->
  pack(-side => 'left', -fill => 'x', -expand =>
  1);
```

- l'option -expand associé à la valeur 1 force le rectangle alloué au widget à remplir l ' espace disponible restant dans la fenêtre ou dans le cadre

Gestion des événements

- Perl/Tk est un langage de programmation *événementielle*, comme beaucoup d'autres langages.
- Lors de l'exécution du programme, l'utilisateur au moyen d'une action sur un composant (clic bouton, validation d'une saisie ..) génére un événement.
- C'est au programmeur de prévoir l'association de cet événement à du code, qui est le plus souvent l'exécution d'une méthode.

Liaison

- En Tk, on appelle **liaison** la relation entre un composant et les événements qu'il intercepte.
- Ainsi le bouton pressé par un clic souris, fait exécuter par défaut le code mis par le programmeur comme valeur de la propriété **`-command`** .

1. `-command => sub { code }` associe l'exécution du code Perl du bloc { }
2. `-command => \&mysub` associe une *référence* à une fonction.
   Attention, il ne faut pas appeler la fonction directement ce qui l'exécuterait immédiatement.
   Le \ crée une référence vers la fonction.

Création d'un widget Bouton

```
$cadre->Button(-text => "load",-command => sub
   {&lire_fic})->pack(-side => 'left');
```

- Le widget Bouton (**Button**) est un des widgets les plus utilisés
- en cliquant sur un bouton, il se passe quelque chose
- Différents type de bouton :
  - bouton traditionnel,
  - cases à cocher,
  - boutons radio

Paramètres du widget Bouton

```
$cadre->Button(-text => "load",-command => sub
   {&lire_fic})->pack(-side => 'left');
```

- l'option -text indique le texte affiché dans le bouton
- l'option -command donne un pointeur vers une fonction qui sera appelée quand le bouton sera pressé

```
#!/usr/bin/perl

use Tk;
$fen = MainWindow->new();
$fen ->minsize('250','60');
$fen -> title("Bonjour");


$texte=$fen -> Label(-textvariable=>\$val);
$texte->pack(       -side=>top,
                    -fill=>both,
                    -expand=>1);

$radio = $fen->Radiobutton(   -text=>'licence',
                              -variable=>\$val,
                              -value=>1);
$radio ->pack(-side=>left);

$radio = $fen ->Radiobutton(  -text=>'maitrise',
                                       -variable=>\$val,
                                       -value=>2);
$radio ->pack(-side=>left);

$radio = $fen ->Radiobutton(  -text=>'autre',
                                       -variable=>\$val,
                                       -value=>3);


$radio ->pack(-side=>left);

$val=1;
```

```
$texte = $fen ->Label(        -text =>"Vos stages de programmation :");

$texte->pack(       -side=>top,
                    -fill=>both,
                    -expand=>1);


$check = $fen ->Checkbutton(-text=>"Php-mysql");
$check -> pack(     -side=>left,
                    -expand=>1);

$check = $fen ->Checkbutton(-text=>"Javascript");

$check->pack(       -side=>left,
          -expand=>1);

$check = $fen ->Checkbutton(-text=>"Java");
$check->pack(       -side=>left,
          -expand=>1);

$check = $fen ->Checkbutton(-text=>"VB");
$check->pack(       -side=>left,
          -expand=>1);

$bouton=$fen -> Button( -text => "Fermer", -command => sub {exit});
$bouton -> pack(-side=>bottom,-expand=>1,fill=>x);

MainLoop;
```

Disposer les boutons selon une grille

- on envoie sur chaque composant une méthode de gestionnaire d'espace **`grid`** (à la place de **`pack`** précédemment)
- Les numéros de (ligne,colonne) qui débutent à (0,0), s'introduisent par ***`grid(-row=>n,-column=>m)`***, où (n,m) sont les coordonnées de la cellule.
- Comme de cette façon chaque composant est affecté à une cellule, il n'est pas nécessaire de les placer dans l'ordre, dans les lignes du code

Exemples

- Ainsi, pour placer un composant de type bouton de commande en rangée 1 et en colonne 2 on écrit :

```
$bouton = $fen -> Button(
        -text=>"Bouton",
        -command=>sub {...})
$bouton -> grid(-row=>1,-column=>2);
```

- Un composant de type Label se code de la même manière :

```
$texte = $fen ->Label(-text=>"mon texte ....")
$texte -> grid(-row=> ..,-column=> ..);
```

- Pour qu'un composant s'étale sur plusieurs cellules, le positionner sur la première cellule à occuper, puis ajouter dans le gestionnaire grid l'option:

```
-columnspan=>n
-rowspan=>n
```

Ajout d'un menu au cadre

```
$b_menu=$cadre->Menubutton(-text => "File",
   -tearoff => 0,
   -relief => 'ridge',
   -menuitems => [['command' => "load",
                   -command => \&lire_fic],
                   ['command' => "Save",
                   -command => \&sauve_fic],
                   ['command' => "Search",
                   -command => \&search_in_fic],
                   ['command' => "Exit",
                   -command =>  sub {exit;}]])
  ->pack(-side => 'left');
```

Création d'un widget Menu

```
$b_menu=$fen->Menubutton(|options…|);
```

- Le widget bouton de menu (`Menubutton`) est un menu déroulant apparaissant sous un bouton quand celui-ci est pressé
- Le menu disparaît lorsque l 'un de ses éléments est sélectionné ou lorsqu'on clique sur un autre endroit de l 'application

Paramètres du widget Menubutton

- L'option -text indique le texte qui sera affiché dans le bouton d ’entrée du menu
- l'option -tearoff associée à 0 spécifie qu ’il n'y aura pas de ligne en pointillés permettant de détacher le menu du bouton
- l'option -relief configure l ’aspect du bouton
- l'option -menuitems est associée à une liste décrivant le contenu de chaque élément du menu

Contenu du widget Menubutton

```
$b_menu=$cadre->Menubutton(-text => "File",
   -menuitems => [['command' => "load",
                   -command => \&lire_fic],
                   ['command' => "Save",
                   -command => \&sauve_fic],
                   ['command' => "Search",
                   -command => \&search_in_fic],
                   ['command' => "Exit",
                   -command =>  sub {exit;}]]);
```

- la valeur associée à menuitems est une liste de liste indiquant l ’ordre d ’apparition des éléments du menu

Paramètrage du MenuButton

```
$b_menu=$cadre->Menubutton(-text => "File",
    -menuitems => [['command' => "load",
                    -command => \&lire_fic],
                    ...])
```

- Chaque sous-liste contient des informations sur les éléments du menu
  - le premier élément de cette sous-liste indique le type de l'entrée du menu (command, radiobutton, checkbutton, cascade…)
  - le deuxième élément indique la chaîne affichée dans le menu

Fonction de rappel sur entrée de menu

- On peut à la suite ajouter des options qui affecteront les entrées définies
  - l ’option -command permet d ’associer une fonction de rappel aux éléments du menu

```
['command' => "load",-command => \&lire_fic]
```

  - ici, via l’option -command, la fonction lire_fic est associée à l’entrée de type command de l’élément du menu dont le libellé est load

Création d'un widget Text

```
$texte = $mw->Text(|options…|);
```

- Le widget `Text` (texte) permet d'afficher du texte…
- Pour insérer du texte

```
$texte->insert('end', "texte à
  insérer") ;
```

  - le premier paramètre de cette fonction est un indice (ici la fin du widget texte) indiquant où insérer le texte donné dans le deuxième paramètre

Widget avec barre de défilement

```
$texte = $mw->Scrolled("Text")->pack(-side => 'bottom',
                                     -fill => 'both',
                                     -expand => 1);
```

- La méthode `Scrolled` appliquée à un widget permet de créer un widget et ses barres de défilement en même temps

Paramètres du widget Text

```
$texte->Text(-setgrid => 'true');
```

- l'option -setgrid active le maillage du widget texte

Indices de texte

- Valeurs des indices
  - « n.m » : format pour spécifier un numéro de ligne et un numéro de caractère dans cette ligne
  - « end » : position juste après la dernière ligne
  - « 1.0 » : le premier caractère de la première ligne

Marqueurs de texte

- Un marqueur de texte permet d ’associer des informations à une ou des parties du texte

```
$w->tag('add', $tag, sprintf("%d.%d", $i, $offset),
   sprintf("%d.%d", $i, $offset+$l));
```

- dans cet exemple,
  - $w est le widget texte auquel on ajoute un marqueur associé à la variable $tag (valant ‘ search ’) à la position donnée par les indices donnés en 3ème et 4ème position

Utilisation des indices/marqueurs

- Insertion de texte

```
$texte->insert(indice,chaîne
  [,liste_marqueurs,chaîne,liste_marqueur
  s…]);
```

- Suppression de texte

```
$texte->delete(début[,fin]);
```

- Capture de texte

```
$texte->get(début[,fin]);
```

- Affichage de texte situé à un indice

```
$texte->see(indice);
```

- Recherche de texte

```
$texte->search([options], motif,
  début[,fin]);
```

  - Les options principales sont : -forward, -backwards, -exact, -regexp, -nocase

```
#!/usr/local/bin

use Tk;
$mw= MainWindow->new;
$mw->title("Recherche avec Expressions régulières");
$cadre = $mw->Frame->pack(-side => 'top',
                          -fill => 'x');
$b_menu=$cadre->Menubutton(-text => "File",
                           -tearoff => 0,
                           -relief => 'ridge',
                           -menuitems => [['command' => "load",
                                           -command => \&lire_fic],
                                          ['command' => "Save",
                                           -command => \&sauve_fic],
                                          ['command' => "Search",
                                           -command => \&search_in_fic],
                                          ['command' => "Exit",
                                           -command =>  sub {exit;}]])-
>pack(-side => 'left');

$cadre->Label(-text => "File : ",
              -relief => 'raised')->pack(-side => 'left',
                                         -anchor => 'w');
$cadre->Entry(-textvariable => \$nom_fic)->pack(-side => 'left',
                                                -anchor => 'w',
                                                -fill => 'x',
                                                -expand => 1);
```

```
$cadre->Button(-text => "load",
               -command => sub {&lire_fic})->pack(-side => 'left',
                                                  -anchor => 'e');
$texte = $mw->Scrolled("Text")->pack(-side => 'bottom',
                                     -fill => 'both',
                                     -expand => 1);
$texte->Text(-setgrid => 'true');
$search_string = '';
$w_string = $mw->Frame()->pack(-side => 'top', -fill => 'x');
$w_string_label = $w_string->Label(-text => 'Search string:', -width =>
13, -anchor => 'w');
$w_string_entry = $w_string->Entry(-textvariable => \$search_string);
$w_string_button = $w_string->Button(-text => 'Highlight');
$w_string_label->pack(-side => 'left');
$w_string_entry->pack(-side => 'left');
$w_string_button->pack(-side => 'left', -pady => 5, -padx => 10);
$w_string->pack(-side => 'top', -fill => 'x');
$w_string_button->configure(-command => [sub {&text_search($texte,
$search_string, 'search')}, $texte]);
$w_string_entry->bind('<Return>' => [sub {shift; &text_search($texte,
$search_string, 'search')}, $texte]);
$w_string->pack(-side => 'top', -fill => 'x');
```

```
if ($mw->depth > 1) {
        text_toggle($texte, ['configure', 'search', -background =>
'SeaGreen4', -foreground => 'white'], 800,
                            ['configure', 'search', -background =>
undef,       -foreground => undef],   200);
      } else {
        text_toggle($texte, ['configure', 'search', -background =>
'black',     -foreground => 'white'], 800,
                            ['configure', 'search', -background =>
undef,       -foreground => undef],   200);
      }
$cadre->Button(-text => "Exit",
               -command => sub {exit;})->pack(-side => 'right');
$cadre->Button(-text => "Save",
               -command => sub {&sauve_fic})->pack(-side => 'right',
                                                   -anchor => 'e');
$mw->Label(-textvariable => \$info,
           -relief => 'ridge')->pack(-side => 'bottom',
                                     -fill => 'x');
MainLoop;
```

```
#--------------------Procédures-----------------------------
sub lire_fic {
    $info="Chargement du fichier '$nom_fic'...";
    $texte->delete("1.0","end");
    if (!open(FIC,"$nom_fic")) {
        $texte->insert("end","ERREUR : Impossible d\'ouvrir
'$nom_fic'\n");
        return;
    }
    while (<FIC>) {
        $texte->insert("end",$_);
    }
    close(FIC);
    $info = "Fichier '$nom_fic' chargé.";
}

sub sauve_fic {

    $info="Sauvegarde du fichier '$nom_fic'...";
    open(FIC,">$nom_fic");
    print FIC $texte->get("1.0","end");
    $info= "Fichier '$nom_fic' sauvegardé.";
}
```

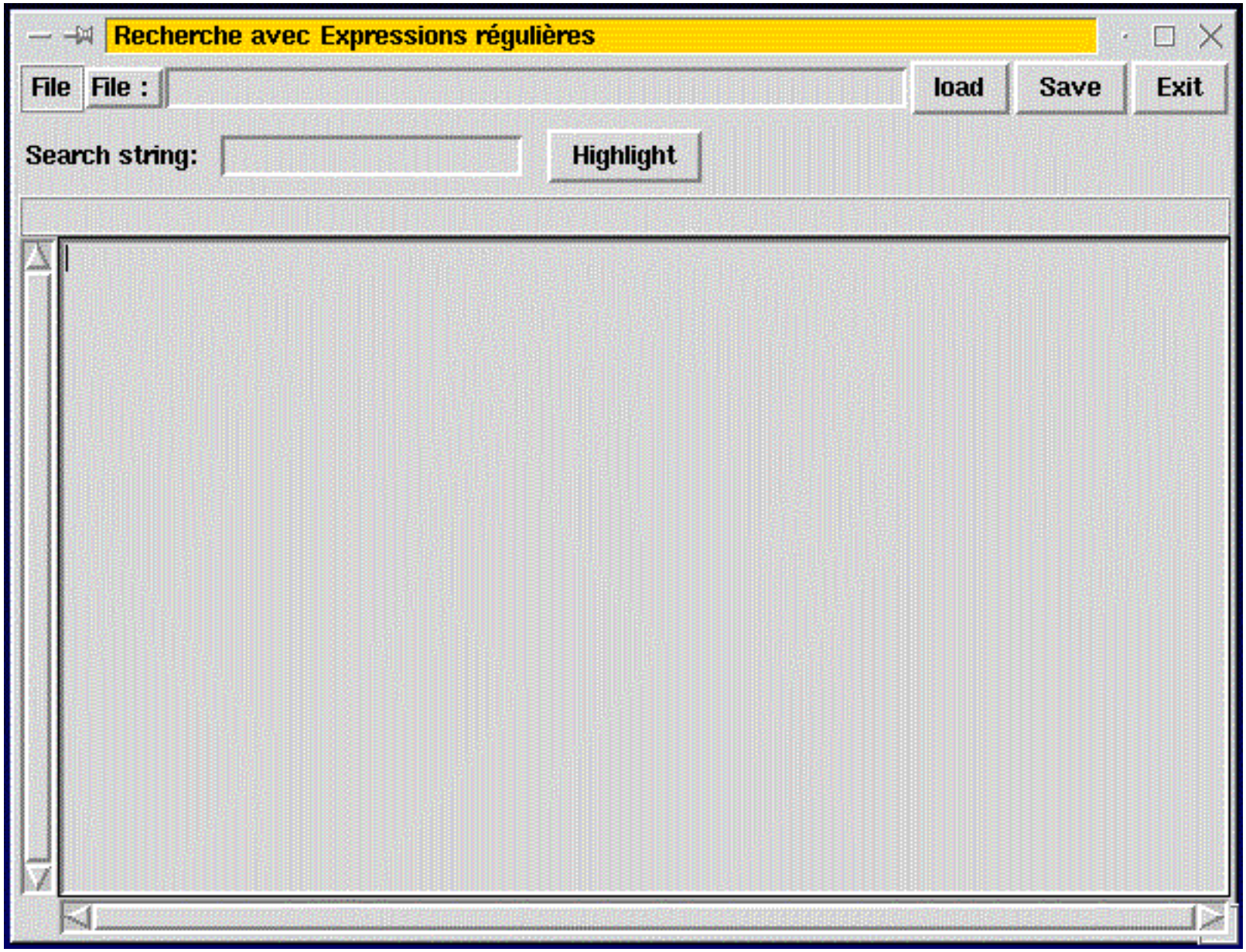
```
sub search_in_fic {

    $info="Recherche dans le fichier '$nom_fic'...";
    &text_search($texte, $search_string, 'search');
    if ($mw->depth > 1) {
        text_toggle($texte, ['configure', 'search', -background =>
'SeaGreen4', -foreground => 'white'], 800,
                            ['configure', 'search', -background =>
undef,       -foreground => undef],   200);
      } else {
        text_toggle($texte, ['configure', 'search', -background =>
'black',     -foreground => 'white'], 800,
                            ['configure', 'search', -background =>
undef,       -foreground => undef],   200);
      }

    $info= "Recherche dans Fichier '$nom_fic' terminée.";
}

sub text_search {

    # The utility procedure below searches for all instances of a given
string in a text widget and applies a given tag
    # to each instance found.
    # Arguments:
    #
```

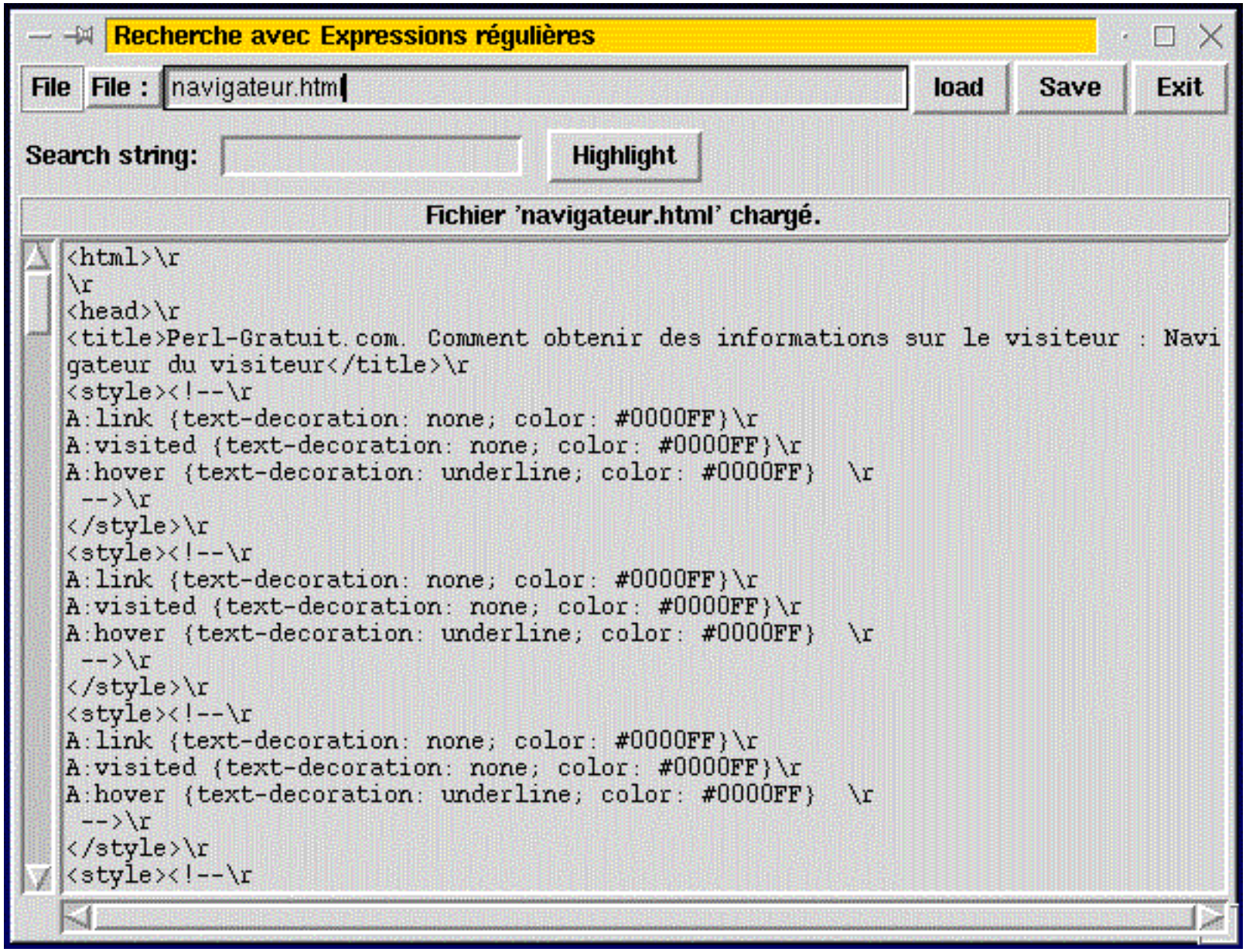
```
# w -    The window in which to search.  Must be a text widget.
    # string -   The string to search for. (regexp and so on...)
    # tag -      Tag to apply to each instance of a matching string.

    my($w, $string, $tag) = @_;

    $w->tag('remove',  $tag, '0.0', 'end');
    (my $num_lines) = $w->index('end') =~ /(\d*)\.\d*/;
    for($i = 1; $i <=$num_lines; $i++) {
         my $line = $w->get("${i}.0", "${i}.1000");
         next if not defined $line or $line !~ /($string)/;
        my $l = length $1;
         $stringRegexp=$1;
         my $offset = 0;
         while (1) {
             my $tmpoffset=0;
             my $index = index $line, $stringRegexp, $tmpoffset;
             last if $index == -1;
             $offset += $index;
             $w->tag('add', $tag, sprintf("%d.%d", $i, $offset),
sprintf("%d.%d", $i, $offset+$l));
             $offset += $l;
             $line = substr $line, $index+$l;
             if ($line=~/($string)/) { $stringRegexp=$1;
                                       $l = length $1;}
```

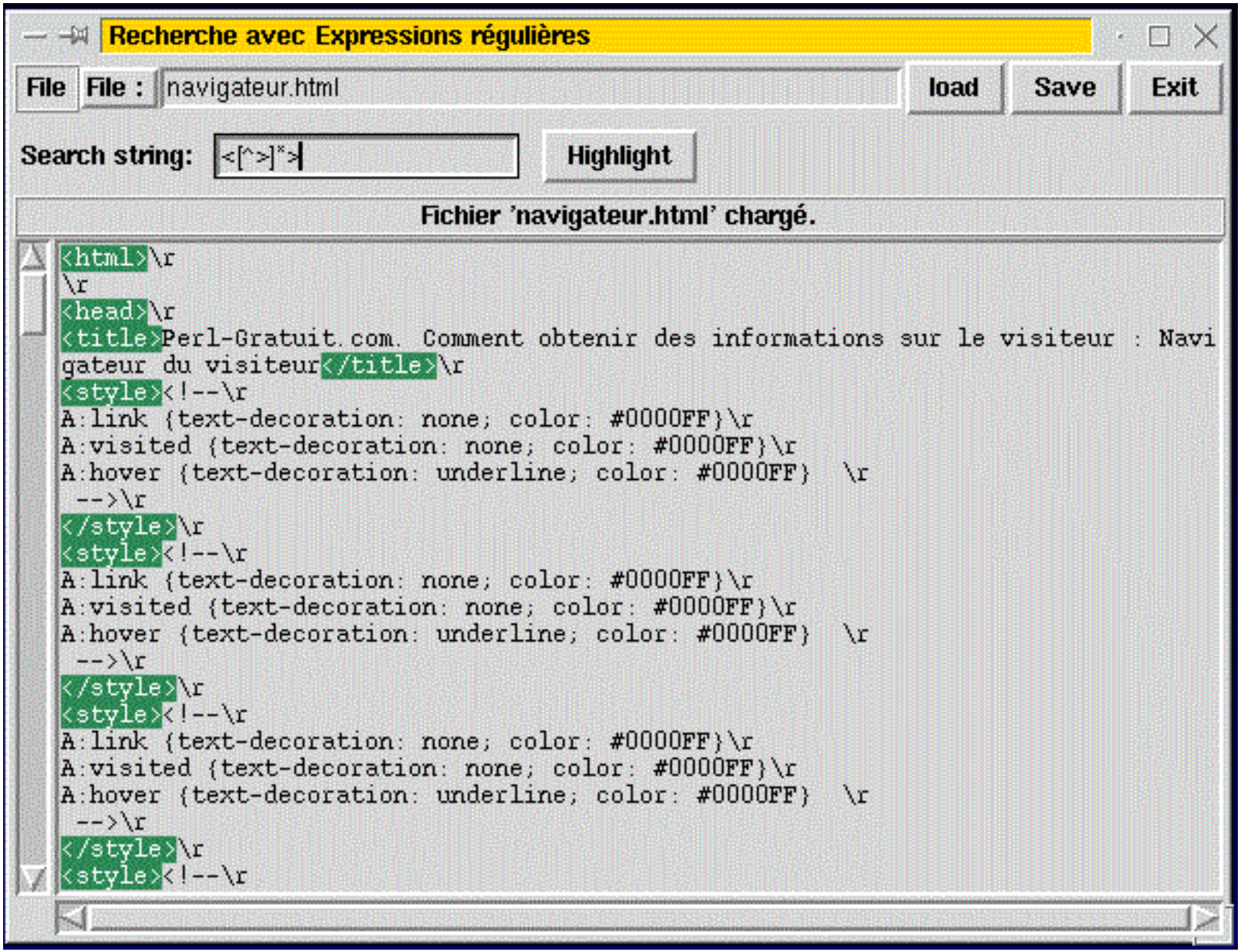
```
        } # whilend
    } # forend

} # end text_search
sub text_toggle {
    # The procedure below is invoked repeatedly to invoke two commands
at periodic intervals.  It normally reschedules itself
    # after each execution but if an error occurs (e.g. because the
window was deleted) then it doesn't reschedule itself.
    # Arguments:
    #
    # w -       Text widget reference.
    # cmd1 -     Reference to a list of tag options.
    # sleep1 -   Ms to sleep after executing cmd1 before executing
cmd2.
    # cmd2 -     Reference to a list of tag options.
    # sleep2 -   Ms to sleep after executing cmd2 before executing cmd1
again.
    my($w, $cmd1, $sleep1, $cmd2, $sleep2) = @_;
# return if not Exists $w;
    $w->tag(@{$cmd1});
    $w->after($sleep1, [sub {text_toggle(@_)}, $w, $cmd2, $sleep2,
$cmd1, $sleep1]);
} # end text_toggle
```

Que fait ce programme

- Ouverture et sauvegarde de fichier
- Recherche de motifs (avec utilisation des expressions régulières) dans le fichier chargé :
    - les zones de textes correspondant au motif de recherche sont mis en valeur par une couleur prédéfinie
- Les différentes options sont disponibles via des menus ou via des boutons insérés dans la fenêtre principale


Recherche avec Expressions régulières
File
File :
load
Save
Exit
Search string:
Highlight


Recherche avec Expressions régulières
File
File :
navigateur.html
load
Save
Exit
Search string:
Highlight
Fichier 'navigateur.html' chargé.
<html>\r
\r
<head>\r
<title>Perl-Gratuit.com. Comment obtenir des informations sur le visiteur : Navi
gateur du visiteur</title>\r
<style><!--\r
A:link {text-decoration: none; color: #0000FF}\r
A:visited {text-decoration: none; color: #0000FF}\r
A:hover {text-decoration: underline; color: #0000FF} \r
 -->\r
</style>\r
<style><!--\r
A:link {text-decoration: none; color: #0000FF}\r
A:visited {text-decoration: none; color: #0000FF}\r
A:hover {text-decoration: underline; color: #0000FF} \r
 -->\r
</style>\r
<style><!--\r
A:link {text-decoration: none; color: #0000FF}\r
A:visited {text-decoration: none; color: #0000FF}\r
A:hover {text-decoration: underline; color: #0000FF} \r
 -->\r
</style>\r
<style><!--\r


Recherche avec Expressions régulières
File
File :
navigateur.html
load
Save
Exit
Search string:
<[^>]*>
Highlight
Fichier 'navigateur.html' chargé.
<html>\r
\r
<head>\r
<title>Perl-Gratuit.com. Comment obtenir des informations sur le visiteur : Navi
gateur du visiteur</title>\r
<style><!--\r
A:link {text-decoration: none; color: #0000FF}\r
A:visited {text-decoration: none; color: #0000FF}\r
A:hover {text-decoration: underline; color: #0000FF} \r
 -->\r
</style>\r
<style><!--\r
A:link {text-decoration: none; color: #0000FF}\r
A:visited {text-decoration: none; color: #0000FF}\r
A:hover {text-decoration: underline; color: #0000FF} \r
 -->\r
</style>\r
<style><!--\r
A:link {text-decoration: none; color: #0000FF}\r
A:visited {text-decoration: none; color: #0000FF}\r
A:hover {text-decoration: underline; color: #0000FF} \r
 -->\r
</style>\r
<style><!--\r