GLib - N-ary Trees

Hai Nam Tran

November 18, 2019

GLib

- GLib est une bibliothèque libre pour le langage C. Elle implémente des structures de données élaborées : `arbres`, `table de hachage` et `listes`.
- On utilise la structure `N-ary Trees` pour l'implantation de l'abre de zpixel !
- https ://developer.gnome.org/glib/stable/glib-N-ary-Trees.html

struct GNode

```c
//The GNode struct and its associated functions provide a N-ary tree data structure, where nodes in the tree can contain arbitrary data.

typedef void* gpointer; //A generic, untyped pointer

struct GNode {
  gpointer data; //contains the actual data of the node.
  GNode    *next; //the node's next sibling
  GNode    *prev; //the node's previous sibling.
  GNode    *parent; //the parent of the GNode
  GNode    *children; //the first child of the GNode
};
```

struct GNode

Exemple : créer un GNode avec un zpixel, en suite, ajouter un fils.

```
1  GNode * g1;
2  struct zpixel * p1;
3  p1 = zpixelCreate(0,0,4);;
4  zpixelInit(p1,255,0,255,0.0);
5  g1 = g_node_new(p1);
6
7  printf("gnode - x: %d \n", ((struct zpixel *)g1->data)->x);
8  printf("gnode - y: %d \n", ((struct zpixel *)g1->data)->y);
9
10 struct zpixel * p2;
11 p2 = zpixelCreate(2,2,2);;
12 zpixelInit(p2,255,0,255,0.0);
13 g_node_append(g1,g2);
14
15 printf(" %d \n", ((struct zpixel *)g1->children->data)->x);
```

GLib fonctions

Description

The GNode struct and its associated functions provide a N-ary tree data structure, where nodes in the tree can contain arbitrary data.

To create a new tree use `g_node_new()`.

To insert a node into a tree use `g_node_insert()`, `g_node_insert_before()`, `g_node_append()` and `g_node_prepend()`.

To create a new node and insert it into a tree use `g_node_insert_data()`, `g_node_insert_data_after()`, `g_node_insert_data_before()`, `g_node_append_data()` and `g_node_prepend_data()`.

To reverse the children of a node use `g_node_reverse_children()`.

To find a node use `g_node_get_root()`, `g_node_find()`, `g_node_find_child()`, `g_node_child_index()`, `g_node_child_position()`, `g_node_first_child()`, `g_node_last_child()`, `g_node_nth_child()`, `g_node_first_sibling()`, `g_node_prev_sibling()`, `g_node_next_sibling()` or `g_node_last_sibling()`.

To get information about a node or tree use `G_NODE_IS_LEAF()`, `G_NODE_IS_ROOT()`, `g_node_depth()`, `g_node_n_nodes()`, `g_node_n_children()`, `g_node_is_ancestor()` or `g_node_max_height()`.

To traverse a tree, calling a function for each node visited in the traversal, use `g_node_traverse()` or `g_node_children_foreach()`.

To remove a node or subtree from a tree use `g_node_unlink()` or `g_node_destroy()`.