Gérald Lelong

Noël Martignoni

G31

Structure de donnée

Compte rendu TP3

Table des matières

1 Présentation du TP

1.1 Description

L'objectif de ce TP est de créer une structure de donnée permettant de stocker une matrice creuse lue depuis un fichier texte dans une table.

2 Programme C

2.1 matrix.h

```
1  #ifndef MATRIX_H
2  #define MATRIX_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  typedef int cell_type;
8
9  typedef struct minor_table_cell_
10 {
11     unsigned int column_number;
12     cell_type value;
13     struct minor_table_cell_ * next;
14 } minor_table_cell_t;
15
16 typedef struct
17 {
18     unsigned int row_number;
19     struct minor_table_cell_ * cell;
20 } major_table_cell_t;
21
22 typedef struct
23 {
24     unsigned int major_table_size;
25     major_table_cell_t * major_table;
26 } matrix_t;
27
28 matrix_t readFromFile(FILE * data_file);
29
30 void addValue(matrix_t * matrix,
31               unsigned int row_number,
32               unsigned int column_number,
```

```
33                cell_type value);
34
35  unsigned int findMajorTableIndex(matrix_t * matrix, unsigned int row_number);
36
37  void insertElemMajorTable(matrix_t * matrix, unsigned int row_number, unsigned int row_insert);
38
39  minor_table_cell_t ** findMinorTableIndex(major_table_cell_t * matrix_cell, unsigned int column_number);
40
41  void insertElemMinorTable(minor_table_cell_t ** minor_cell, unsigned int column_insert, cell_type value);
42
43  void printTable(matrix_t * matrix);
44
45  void freeTable(matrix_t * matrix);
46
47  #endif
```

Listing 2.1 – ../matrix.h

2.2 matrix.c

```
1   #include "matrix.h"
2
3   #define BUFFER_SIZE (1024)
4
5   matrix_t readFromFile(FILE * data_file)
6   {
7       unsigned int major_table_size = 0;
8       unsigned int row_number=0;
9       unsigned int column_number=0;
10      cell_type value;
11      matrix_t matrix;
12      char buffer[BUFFER_SIZE];
13
14      /* Read major table size */
15      fgets(buffer, BUFFER_SIZE, data_file);
16      sscanf(buffer, "%u", &major_table_size);
17      matrix.major_table = malloc(sizeof(major_table_cell_t) * major_table_size);
18      matrix.major_table_size = 0;
19
20      while(fgets(buffer, BUFFER_SIZE, data_file))
21      {
22          sscanf(buffer, "%u %u %d", &row_number,
23                                     &column_number,
24                                     &value);
25          addValue(&matrix, row_number, column_number, value);
26      }
```

```
27
28      return matrix;
29  }
30
31  void addValue(matrix_t * matrix,
32                unsigned int row_number,
33                unsigned int column_number,
34                cell_type value)
35  {
36      unsigned int major_table_index = 0;
37      minor_table_cell_t ** cell;
38
39      major_table_index = findMajorTableIndex(matrix, row_number);
40
41      cell = findMinorTableIndex(&(matrix->major_table[major_table_index]), column_number);
42      insertElemMinorTable(cell, column_number, value);
43  }
44
45  unsigned int findMajorTableIndex(matrix_t * matrix, unsigned int row_number)
46  {
47      int mean_index = 0;
48      int min_index = 0;
49      int max_index = matrix->major_table_size-1;
50      int is_found = 0;
51
52      while((min_index <= max_index) && !is_found)
53      {
54          mean_index = (min_index + max_index) / 2;
55
56          if (matrix->major_table[mean_index].row_number == row_number)
57              is_found=1;
58          else
59          {
60              if (matrix->major_table[mean_index].row_number > row_number)
61                  max_index = mean_index-1;
62              else
63                  min_index = mean_index+1;
64          }
65      }
66
67      if (!is_found)
68      {
69          if (mean_index != 0) mean_index++;
70          insertElemMajorTable(matrix, row_number, mean_index);
71      }
72
73      return mean_index;
```

```
74  }
75
76  void insertElemMajorTable(matrix_t * matrix, unsigned int row_number, unsigned int row_insert)
77  {
78      int i;
79
80      for (i = matrix->major_table_size ; i > row_insert ; i--)
81      {
82          matrix->major_table[i] = matrix->major_table[i-1];
83      }
84      matrix->major_table[row_insert].cell = NULL;
85      matrix->major_table[row_insert].row_number = row_number;
86      matrix->major_table_size++;
87  }
88
89  minor_table_cell_t ** findMinorTableIndex(major_table_cell_t * matrix_cell, unsigned int column_number)
90  {
91      minor_table_cell_t ** prev = &(matrix_cell)->cell;
92      minor_table_cell_t * current = matrix_cell->cell;
93
94      while (current != NULL && current->column_number < column_number)
95      {
96          prev = &(current)->next;
97          current = *(prev);
98      }
99
100     return prev;
101 }
102
103 void insertElemMinorTable(minor_table_cell_t ** minor_cell, unsigned int column_insert, cell_type value)
104 {
105     minor_table_cell_t * newColumn = malloc(sizeof(minor_table_cell_t));
106     newColumn->column_number = column_insert;
107     newColumn->value = value;
108     newColumn->next = (*minor_cell);
109     (*minor_cell) = newColumn;
110 }
111
112 void printTable(matrix_t * matrix)
113 {
114     struct minor_table_cell_ * cell = NULL;
115     int i;
116
117     printf("\n********** Matrix **********\n\n");
118     for (i = 0 ; i < matrix->major_table_size ; i++)
119     {
120         printf("row : %d\n", matrix->major_table[i].row_number);
```

```
121             cell = matrix->major_table[i].cell;
122
123             while (cell != NULL)
124             {
125                 printf("    col : %d value : %d\n", cell->column_number, cell->value);
126                 cell = cell->next;
127             }
128         }
129         printf("\n**************************\n\n");
130     }
131
132     void freeTable(matrix_t * matrix)
133     {
134         struct minor_table_cell_ * cell = NULL;
135         struct minor_table_cell_ * cell_to_delete = NULL;
136         int i;
137
138         for (i = 0 ; i < matrix->major_table_size ; i++)
139         {
140             cell = matrix->major_table[i].cell;
141
142             while (cell != NULL)
143             {
144                 cell_to_delete = cell;
145                 cell = cell->next;
146                 free(cell_to_delete);
147             }
148         }
149
150         free(matrix->major_table);
151     }
```

Listing 2.2 – ../matrix.c

3 Compilation et tests

3.1 Makefile

```
1  ##############################################################################
2  ##                          Makefile generique                              ##
3  ##############################################################################
4
5  # Nom du projet
6  TARGET   = tp3
7
8  # Compilateur
9  CC       = gcc
10
11 # Options de compilation
12 CFLAGS   = -g -ansi -Wall -pedantic
13
14 # Editeur de liens
15 LINKER   = gcc -o
16
17 # Options de l'editeur de liens
18 LFLAGS   = -Wall
19
20 # Dossier des sources, des binaires et des fichiers de compilation
21 SRCDIR   = .
22 OBJDIR   = ./obj
23 BINDIR   = ./bin
24
25 SOURCES  = $(wildcard $(SRCDIR)/*.c)
26 INCLUDES = $(wildcard $(SRCDIR)/*.h)
27 OBJECTS  = $(SOURCES:$(SRCDIR)/%.c=$(OBJDIR)/%.o)
28 rm       = rm -frv
29
30 $(BINDIR)/$(TARGET): $(OBJECTS)
31         @$(LINKER) $@ $(LFLAGS) $(OBJECTS)
32         @echo "Linking complete!"
```

```
33
34 -include $(OBJECTS:.o=.d)
35
36 $(OBJECTS): $(OBJDIR)/%.o : $(SRCDIR)/%.c
37         @$(CC) $(CFLAGS) -c $< -o $@
38
39         @$(CC) -MM $(CFLAGS) $(SRCDIR)/$*.c > $(OBJDIR)/$*.d
40         @mv -f $(OBJDIR)/$*.d $(OBJDIR)/$*.d.tmp
41         @sed -e 's|.*:|$(OBJDIR)/$*.o:|' < $(OBJDIR)/$*.d.tmp > $(OBJDIR)/$*.d
42         @sed -e 's/.*://' -e 's/\\$$//' < $(OBJDIR)/$*.d.tmp | fmt -1 | \
43           sed -e 's/^ *//' -e 's/$$/:/' >> $(OBJDIR)/$*.d
44         @rm -f $(OBJDIR)/$*.d.tmp
45         @echo "Compiled "$<" successfully!"
46
47
48 # Creation des dossiers utiles a la compilation (si necessaire)
49 mkdirs:
50         mkdir -p $(OBJDIR)
51         mkdir -p $(BINDIR)
52
53 # Generation du compte rendu
54 texcr: $(TARGET).tex $(SRCDIR)/*.c $(SRCDIR)/*.h
55         #valgrind --leak-check=full --log-file=reportincludes/valgrind-report \
56         #       $(BINDIR)/$(TARGET) datafiles/sorted.dat datafiles/unsorted.dat
57         #$(BINDIR)/$(TARGET) datafiles/sorted.dat \
58         #       datafiles/unsorted.dat > reportincludes/output 2>&1
59         #iconv -f UTF-8 -t ISO-8859-1 reportincludes/output -o \
60         #       reportincludes/output.tmp && \
61         #       mv reportincludes/output.tmp reportincludes/output
62         ../mkcr ./sourcefiles.tex $(SRCDIR)/*.c $(SRCDIR)/*.h
63         pdflatex -interaction=nonstopmode $(TARGET).tex
64
65 # Nettoyages
66 .PHONY: texclean
67 texclean:
68         @$(rm) *.aux *.bbl *.blg *.log
69         @echo "Latex cleanup complete!"
70
71 .PHONY: clean
72 clean: texclean
73         @$(rm) $(OBJDIR)/*
74         @$(rm) sourcefiles.tex
75         @echo "Cleanup complete!"
76
77 .PHONY: mrproprer
78 mrproprer: clean
79         @$(rm) $(BINDIR)/$(TARGET)
```

```
80          @$(rm) $(TARGET).pdf
81          @echo "Executable removed!"
```

Listing 3.1 – ../Makefile

3.2 Jeux de test

3.2.1 Ajout d'une valeur dans une matrice vide

```
1 5
```

Listing 3.2 – ../empty_matrix.dat

```
1  #include "matrix.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main()
6  {
7      FILE* file=fopen("empty_matrix.dat","r");
8      matrix_t matrix=readFromFile(file);
9      fclose(file);
10     addValue(&matrix,23,44,90);
11     printTable(&matrix);
12     freeTable(&matrix);
13
14     return 0;
15 }
```

Listing 3.3 – ../tests/insertion_matrice_vide.c

```
1  ********** Matrix **********
2
3  row : 23
4      col : 44 value : 90
5
6  ****************************
```

Listing 3.4 – Sortie

La valeur est bien ajoutée à la matrice.

3.2.2 Lecture d'un fichier vide

```c
#include "matrix.h"
#include <stdio.h>
#include <stdlib.h>

int main()
{
    FILE* file=fopen("empty_file.dat","r");
    matrix_t matrix=readFromFile(file);
    fclose(file);
    freeTable(&matrix);

    return 0;
}
```

Listing 3.5 – ../tests/lecture_fichier_vide.c

Aucune erreur de segmentation lors de l'exécution.

3.2.3 Lecture d'un fichier dans le désordre

```
4
12 14 -51
5 8 -4
25 5 311
55 31 -45
12 23 111
55 21 -123
12 2 256
25 18 79
```

Listing 3.6 – ../matrix.dat

```c
#include "matrix.h"
#include <stdio.h>
#include <stdlib.h>

int main()
{
    FILE* file=fopen("matrix.dat","r");
    matrix_t matrix=readFromFile(file);
    fclose(file);
    printTable(&matrix);
    freeTable(&matrix);

    return 0;
}
```

Listing 3.7 – ../tests/lecture_desordre.c

```
1       ********** Matrix **********
2
3   row : 5
4       col : 8 value : -4
5   row : 12
6       col : 2 value : 256
7       col : 14 value : -51
8       col : 23 value : 111
9   row : 25
10      col : 5 value : 311
11      col : 18 value : 79
12  row : 55
13      col : 21 value : -123
14      col : 31 value : -45
15
16  ****************************
```

Listing 3.8 – Sortie

La matrice est conforme au fichier d'entrée.