```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3: /*
 4:  * FIFO Listes chainÃ©es
 5:  * Slide 18 ou 19.
 6:  *
 7:  * Start with structs.
 8:  * Them simple main.
 9:  * Then simple functions !!!
10:  * Then complex ones.
11:  * Then go back to main.
12:  */
13: typedef unsigned char val_type;
14:
15: struct cellule {
16:         val_type val;
17:         struct cellule *suiv;
18: };
19: struct fifo {
20:         struct cellule *tete;
21:         struct cellule *queue;
22: };
23:
24: int fifo_vide(struct fifo file) {
25:         return (file.tete == NULL);
26: }
27:
28: int fifo_pleine(struct fifo file) {
29:         return 0;
30: }
31:
32: void print_fifo(struct fifo file) {
33:         struct cellule *tmp = file.tete;
34:
35:         printf("-----------\n");
36:         printf("fifo_pleine ? %d, fifo_vide ? %d ::::: 
    ", fifo_pleine(file), fifo_vide(file));
37:         /*printf("Fifo : ");*/
38:         while (tmp != NULL) {
39:                 printf("%c,", tmp->val);
40:                 tmp = tmp->suiv;
41:         }
42:         /*printf("\nFifo est vide ? %d, Fifo est pleine 
    ? %d\n",
43:          * fifo_vide(file), fifo_pleine(file));*/
```



```
44:         printf("\n-----------\n\n");
45: }
46:
47:
48: void init_fifo(struct fifo *ptr_file)
49: {
50:         ptr_file->tete = NULL;
51:         ptr_file->queue = NULL;
52: }
53:
54: val_type first(struct fifo file)
55: {
56:         return file.tete->val;
57: }
58:
59: val_type get(struct fifo *ptr_file)
60: {
61:         val_type v = ptr_file->tete->val;
62:
63:         struct cellule *tmp = ptr_file->tete;
64:
65:         ptr_file->tete = ptr_file->tete->suiv;
66:         if (ptr_file->tete == NULL)
67:                 ptr_file->queue = NULL;
68:
69:         free(tmp);
70:
71:         return v;
72: }
73:
74: // put = ajouter en queue !
75: void put(struct fifo *ptr_file, val_type val) {
76:         struct cellule *tmp = malloc(sizeof(struct 
    cellule));
77:         tmp->val = val;
78:         tmp->suiv = NULL;
79:
80:         if (ptr_file->tete == NULL)
81:                 ptr_file->tete = tmp;
82:         else
83:                 ptr_file->queue->suiv = tmp;
84:
85:         ptr_file->queue = tmp;
86: }
87:
```

```
 88: int simple_main() {
 89:         struct fifo f;
 90:         init_fifo(&f);
 91:
 92:         print_fifo(f);
 93:
 94:         val_type x = 'W';
 95:         put(&f, x);
 96:         print_fifo(f);
 97:
 98:         printf("Fifo vide 1 = %d ?\n", fifo_vide(f));
 99:
100:         x = get(&f);
101:         print_fifo(f);
102:
103:         printf("Fifo vide 2 = %d ?\n", fifo_vide(f));
104:
105:         return 0;
106: }
107:
108: int main_complex() {
109:         struct fifo f;
110:         init_fifo(&f);
111:
112:         for (val_type c = 'a'; c < 'a' + 26; c++) {
113:                 print_fifo(f);
114:                 put(&f, c);
115:         }
116:
117:         while (!fifo_vide(f)) {
118:                 printf("RÃ©cupÃ©rÃ©: %c\n", get(&f));
119:                 // get(&f);
120:                 print_fifo(f);
121:         }
122:
123:         return 0;
124: }
125: int main(void) {
126:         simple_main();
127:         return 0;
128: }
129:
130: /* //UNUSED because get function is there !!!
131: void sup_tete(struct fifo * ptr_file) {
132:         struct cellule * tmp = ptr_file->tete ;
```



```
133:
134:         if( ptr_file->tete == ptr_file->queue) {
135:                 ptr_file->tete  = NULL;
136:                 ptr_file->queue = NULL;
137:         } else {
138:                 ptr_file->tete = ptr_file->tete->suiv ;
139:         }
140:         free(tmp);
141: }
142: */
143: //UNUSED !!!
144: void sup_tete_fifo(struct fifo * ptr_file) {
145:         struct cellule * tmp = ptr_file->tete ;
146:
147:         if( ptr_file->tete == ptr_file->queue) {
148:                 ptr_file->tete  = NULL;
149:                 ptr_file->queue = NULL;
150:         } else {
151:                 ptr_file->tete = ptr_file->tete->suiv ;
152:         }
153:         free(tmp);
154: }
155:
156: //DANS DS... pas donner aux eleves
157: int delete_queue(struct fifo * f ) {
158:         if(f==NULL)       return 0; //Nothing deleted,
     no fifo
159:         if(f->tete==NULL) return 0; //Nothing deleted,
     fifo empty
160:
161:         if(f->tete->suiv == NULL) { //Only one cell,
     update tete et queue
162:                 sup_tete_fifo(f);
163:                 return 1;               //deleted a
     value.
164:         }
165:         struct cellule * tmp = f->tete;
166:         while(tmp->suiv->suiv != NULL)
167:                 tmp=tmp->suiv;
168:         //tmp pointe sur l'avant dernier
169:         free(tmp->suiv);  //free last
170:         tmp->suiv = NULL; //update pointer
171:         f->queue = tmp;   //update tail
172:         return 1;
173: }
```