



CHOIX TECHNIQUES

Projet Informatique 1A

Equipe :

DELRUE Pierre

LOUET Thibault

LUMIA Yann

PERSON Victor

15 avril 2016

Table des matières

1	Introduction	2
1.1	Introduction du sujet	2
1.2	Conditions/Pré-requis	2
2	Structures de données	3
2.1	Structure de personne et structure de planète	3
2.2	Structure de liste	3
2.3	Structure de tas	3
3	Algorithme d'affectation	4
4	Découpage en modules	9
4.1	conversion.h	9
4.2	listPlanete.h	10
4.3	listPersonne.h	11
4.4	tas.h	12
5	Choix du langage de programmation	14

1 Introduction

1.1 Introduction du sujet

Une agence de tourisme spatial propose différentes croisières. Le but de ce projet est de distribuer les différentes places disponibles selon le choix des clients en fonction de leur indice de priorité.

Chaque personne a deux souhaits choisis parmi des croisières pré-organisées plus un choix de croisière libre. Si les croisières pré-organisées ne sont plus disponibles, le choix libre sera attribué par défaut. Chaque souhait est constitué de 6 planètes choisies parmi six zones.

L'attribution des croisières (organisées ou libres) se fait d'abord pour les personnes avec une croisière pré-organisée en premier souhait indépendamment de la priorité. Par exemple, une personne avec une priorité de 190 pour une croisière libre sera traitée après une personne de priorité 100 mais dont le souhait est une croisière organisée.

Le programme final prend donc en entrée un tableau contenant le nom, prénom, la priorité et les souhaits de chaque personne, un tableau contenant les différentes croisières possibles et un tableau contenant les contraintes des planètes.

1.2 Conditions/Pré-requis

1. Tous les parcours pré-conçus respectent les contraintes ;
2. Les personnes associées à un parcours ne respectant pas les contraintes perdent toute priorité ;
3. Il est impossible pour 2 planètes distinctes d'une même zone d'être liées par des contraintes ;
4. Si un client fait, comme premier choix, un choix libre, alors son deuxième choix sera nécessairement libre lui aussi.

2 Structures de données

2.1 Structure de personne et structure de planète

Les structures de personne et de planète sont des structures mixtes ayant différents attributs. Pour les personnes, la structure contient le nom, le prénom, la priorité, les deux id des choix, les tableaux de choix plus un tableau de parcourt libre. Pour les planète, la structure contient le nom de la planète et le nombre de places restantes.

2.2 Structure de liste

La structure de liste, de part sa taille dynamique, permet un traitement de données de même type mais dans un nombre variable. Ce qui est impossible avec un tableau qui a une taille fixe. Dans notre cas, il y aura un type liste de personnes et un type liste de planètes.

Ici, on doit traiter des personnes qui pourront avoir une même priorité. Les stocker dans une liste permettra d'optimiser l'attribution des choix. Par exemple, si la première personne n'a pas ses choix attribués et que l'on est obligé de donner des planètes au hasard, il nous sera possible de traiter d'abord les autres personnes de même priorité pour ne pas bloquer un de leur choix en attribuant aléatoirement une planète à celui d'avant. De même, pour les planètes, il y a un nombre variable de planètes entre les zones d'où la nécessité de les stocker dans des listes.

2.3 Structure de tas

La structure de tas s'est imposée comme structure pour le traitement des personnes en fonction de leur priorité. En effet, le prédicat de base de la structure, la clef du père est plus grande que la clef de chacun de ses fils, permet immédiatement de récupérer l'objet dont la clef est la plus grande.

Dans notre cas, la clef des noeuds sera la priorité de chaque personne et le noeud contiendra le nom, le prenom, deux entiers contenant l'id de chacun des deux souhaits formulés ainsi que trois tableaux qui pourront contenir 2 choix de croisière organisée et la croisière libre par défaut.

3 Algorithme d'affectation

NB : dans la suite,

- *partie 1 des contraintes désigne les destinations qui en impliquent une autre ;*
- *partie 2 des contraintes désigne les destinations impliquées par d'autres ;*

Algorithm 1: Algorithme d'affectation en pseudo-code - partie 1/4

Données: Tableau de voyageurs TdV
Resultat: Tableau traité

```
1 Initialisation;
2  $A \leftarrow \emptyset$ ;
3 parcours de TdV faire
4   si choix1  $\neq$  libre alors
5     | Insérer dans A à indice(priorité + 1000);
6   sinon
7     | Insérer dans A à indice(priorité);
8 Corps de l'algorithme;
9  $L \leftarrow \emptyset$ ;
10 parcours de A, dans l'ordre décroissant d'indice faire
11    $C \leftarrow$  suivant de A;
12   si  $L = \emptyset$  ou prioté du dernier élément = priorité(C) alors
13     | Insérer C dans L;
14     | Supprimer C de A;
15   sinon
16     parcours de L faire
17        $C \leftarrow$  suivant de L;
18       si choix1 de C disponible alors
19         | Affecter C à la liste correspondant au choix1;
20         | Supprimer C de L;
21     parcours de L faire
22        $C \leftarrow$  suivant de L;
23       si choix2 de C disponible alors
24         | Affecter C à la liste correspondant au choix2;
25         | Supprimer C de L;
26       sinon
27         | Affecter C à la liste de choix libre;
28         | Supprimer C de L;
29   |  $L \leftarrow \emptyset$ ;
```

Algorithm 2: Algorithme d'affectation en pseudo-code - partie 2/4

```
30 pour chaque structure des voyages pré-organisée, A faire
31   pour  $i \leftarrow 0$  à  $nbZones - 1$  faire
32      $N \leftarrow nbPlaces$  disponibles dans la zone  $i$ ;
33      $T \leftarrow \emptyset$ ;
34      $L \leftarrow \emptyset$ ;
35     parcours de A, dans l'ordre décroissant d'indice faire
36        $C \leftarrow$  suivant de A;
37       si  $L = \emptyset$  prioté du dernier élément = priorité(C) alors
38         Insérer C dans L;
39         Supprimer C de A ;
40       sinon
41         parcours de L, dans l'ordre décroissant d'indice faire
42            $C \leftarrow$  suivant de A;
43           si choix de la zone i de C possible alors
44             Affecter son choix zone  $i$  à sa destination zone  $i$ ;
45           sinon
46             Insérer C dans T;
47          $L \leftarrow \emptyset$ ;
48       parcours de T, dans l'ordre décroissant d'indice faire
49          $C \leftarrow$  suivant de T;
50         Affecter son choix zone  $i$  à la 1re destination zone  $i$  possible;
```

Algorithm 3: Algorithme d'affectation en pseudo-code - partie 3/4

```
51  $L \leftarrow \emptyset, T_1 \leftarrow \emptyset, T_2 \leftarrow \emptyset;$ 
52 parcours de la sstructure des parcours libre de A, dans l'ordre décroissant d'indice faire
53   C  $\leftarrow$  suivant de A;
54   si C ne respecte pas les contraintes alors
55     Insérer C dans  $T_1$ ;
56   Supprimer C de A;
57   sinon
58     pour chaque contrainte faire
59       P  $\leftarrow$  prochaine planète de partie 2 des contraintes;
60       Q  $\leftarrow$  planète impliquant P;
61       si  $P \in \text{choix}(C)$  et  $Q \notin \text{choix}(C)$  et  $\text{nbPlaces}(Q) > \text{nbPlaces}(P) - 1$  alors
62         Insérer C dans  $T_2$ ;
63       Supprimer C de A;
64     // début allocation destinations
65     si C non inséré dans  $T_2$  alors
66       pour  $i \leftarrow 1$  à  $\text{nbZones} - 1$  faire
67         si destination de zone i de C vide alors
68           P  $\leftarrow$  planète zone i choisie par C;
69           si P est dans partie 1 des contraintes alors
70             Q  $\leftarrow$  planète impliquée par P;
71             si P ou Q non libre alors
72               Insérer C dans  $T_2$ ;
73             Supprimer C de A;
74           sinon
75             Affecter P et Q à la destination de C;
76         sinon si P est dans partie 2 des contraintes alors
77           Q  $\leftarrow$  planète impliquant P;
78           si  $Q \in \text{choix}(C)$  alors
79             Affecter P et Q à la destination de C;
80           sinon si  $\text{nbPlaces}(P) - 1 < \text{nbPlaces}(Q)$  alors
81             Insérer C dans  $T_2$ ;
82             Supprimer C de A;
83           sinon
84             Affecter P à destination de C;
85         sinon
86           si P est libre alors
87             Affecter P à destination de C;
88           sinon
89             Insérer C dans  $T_2$ ;
90             Supprimer C de A;
```

Algorithm 4: Algorithme d'affectation en pseudo-code - partie 4/4

```
90  $T \leftarrow \emptyset$ ;  
91 Insérer  $T_1$  et  $T_2$  dans  $T$ ;  
92 parcours de  $T$ , dans l'ordre décroissant d'indice faire  
93    $C \leftarrow$  suivant de  $T$ ;  
94   pour  $i \leftarrow 1$  à  $nbZones - 1$  faire  
95     si zone  $i$  pas déjà affectée alors  
96       si  $P$  libre alors  
97         si  $P$  n'a aucune contrainte alors  
98           Affecter  $P$  à destination de  $C$ ;  
99         sinon  
100           si  $P$  dans Partie 1 des contraintes alors  
101              $Q \leftarrow$  planète impliquée par  $P$ ;  
102             si  $Q$  libre alors  
103               si zone de  $Q$  non affectée ou  $choix(C) = Q$  alors  
104                 Affecter  $P$  et  $Q$  à destination de  $C$ ;  
105             sinon  
106                $Q \leftarrow$  planète impliquant  $P$ ;  
107               si  $nbPlaces(P) - 1 \geq nbPlaces(Q)$  alors  
108                 Affecter  $P$  à destination de  $C$ ;  
109     si zone  $i$  pas déjà affectée alors  
110       parcours de autres planètes faire  
111          $P \leftarrow$  planète suivante;  
112         si  $P$  libre et  $P \notin$  partie 1 des contraintes alors  
113           si  $P$  dans partie 2 des contraintes alors  
114              $Q \leftarrow$  planète impliquant  $P$ ;  
115             si  $nbPlaces(P) - 1 \geq nbplaces(Q)$  alors  
116               Affecter  $P$  à destination de  $C$ ;  
117               fin parcours(autres planètes);  
118             si  $choix(C) = Q$  alors  
119               Affecter  $P$  et  $Q$  à destination de  $C$ ;  
120               fin parcours(autres planètes);  
121           sinon  
122             Affecter  $P$  à  $C$ ;  
123             fin parcours(autres planètes);  
124         sinon si  $P$  libre et  $P \in$  partie 1 des contraintes alors  
125            $Q \leftarrow$  planète impliquée par  $P$ ;  
126           si  $Q$  libre et zone de  $Q$  non affectée alors  
127             Affecter  $Q$  et  $P$  à destination de  $C$ ;  
128             fin parcours(autres planètes);  
129   * // insérer code page 8  
130 Fin de l'Algorithme
```

Algorithm 5: (*)

```
129 pour chaque planète dans partie 1 des contraintes faire
130   P ← planète suivante;
131   Q ← planète impliquée par P;
132   si zone de Q non affectée à C alors
133     parcours de autres planètes de la zone de Q faire
134       R ← planète suivante;
135       si R libre alors
136         Affecter R à destination de C;
```

Remarque : () permet de régler le cas où il n'y a plus assez de place autre part que dans une planète qui en implique une autre qui n'a pas été choisie par C (pas besoin de tester les contraintes sur R grâce à la condition 3 (cf 1.2. Conditions/Pré-requis))*

4 Découpage en modules

4.1 conversion.h

```
1  #ifndef CONVERSION_H_INCLUDED
2  #define CONVERSION_H_INCLUDED
3
4  #include "listePersonne.h"
5  #include "listePlanete.h"
6  #include "tas.h"
7
8  /* @requires : f1 et f2 deux flux sur les fichiers des personnes
9     (f1 = flux sur le tableau de priorites et f2 = flux sur le tableau de
10     souhaits)
11     @assigns : nothing
12     @ensures : retourne le tas ou sont ranges toutes les personnes par
13     ordre de priorite */
14  tas convertir_t_personne(FILE * f1, FILE * f2);
15
16  /* @requires : fd un flux sur le fichier des zones
17     @assigns : nothing
18     @ensures : retourne le tableau de listes de planete */
19  liste_planete* convertir_t_planete(FILE * fd);
20
21  /* @requires : f un flux sur le fichier des contraintes, sizetab le
22     nombre de lignes du fichier
23     @assigns : nothing
24     @ensures : retourne le tableau des contraintes */
25  char*** convertir_contrainte(FILE * f, int sizetab) ;
26
27  /* @requires : une personne et un flux sur un fichier
28     @assigns : rajoute une ligne au fichier
29     @ensures : la ligne a ete ajoutee */
30  void ajout_ligne(personne, FILE * ) ;
31
32  /* @requires : un flux de fichier
33     @assigns : nothing
34     @ensures : retourne le nombre de lignes du fichier */
35  int nombreligne(FILE * f) ;
36
37 #endif /* CONVERSION_H_INCLUDED */
```

4.2 listPlanete.h

```
1  #ifndef LISTPLA_H
2  #define LISTPLA_H
3
4  typedef struct planete_base {
5      char* nom; /* nom de la planete */
6      int nbPlaces; /* nombre de places restantes */
7  } * planete;
8
9  typedef struct liste_planete_base {
10     planete val;
11     struct liste_planete_base * next;
12 } * liste_planete;
13
14 /* @requires : nothing
15    @assigns : nothing
16    @ensures : retourne une planete */
17 planete creer_planete();
18
19 /* @requires : nothing
20    @assigns : nothing
21    @ensures : retourne une liste de planete vide */
22 liste_planete creer_liste_planete();
23
24 /* @requires : une liste de planete valide
25    @assigns : nothing
26    @ensures : retourne 1 si la liste est vide et 0 sinon */
27 int liste_planete_vide(liste_planete);
28
29 /* @requires : un pointeur sur liste de planete valide et une planete
30    valide, non presente dans la liste
31    @assigns : la liste de planete
32    @ensures : la planete a ete inseree en tete de liste */
33 void inserer_liste_planete(liste_planete*, planete);
34
35 /* @requires : tableau de liste_planete tabListPla valide, verifiant
36    len(tabListPla) == tabSize
37    @assigns : nothing
38    @ensures : si namePla est le nom d'une planete contenue dans
39                tabListPla, retourne le nombre de places restantes de cette planete
40                ; sinon, retourne -1 */
41 int nb_places_planete(char* namePla, liste_planete* tabListPla, int
42     tabSize);
43
44 #endif /* LISTPLA_H */
```

4.3 listPersonne.h

```
1  #ifndef LISTPER_H
2  #define LISTPER_H
3
4  typedef struct personne_base {
5      int priorite;
6      char* nom;
7      char* prenom;
8      int id1; /* identifiant choix 1 */
9      int id2; /* identifiant choix 2 */
10     char* tabChxOrg1[6]; /* tableau correspondant au 1er choix de voyage
        pre-organise de la personne */
11     char* tabChxOrg2[6]; /* tableau correspondant a son 2nd voeu de voyage
        de type pre-organise */
12     char* tabChxLib[6]; /* tableau contenant les choix de la personne dans
        le cas d'un parcours libre */
13     int assigned[6]; /* tableau qui indique les zones assignees de maniere
        definitive au voyageur */
14     int chxFin; /* entier qui indique le choix attribue a la fin a l'
        individu (1 : chxOrg1 ; 2 : chxOrg2 ; 3 : chxLib) */
15 } * personne;
16
17 typedef struct liste_personne_base {
18     personne val;
19     struct liste_personne_base * next;
20 } * liste_personne;
21
22 /* @requires : nothing
23    @assigns : nothing
24    @ensures : retourne une personne */
25 personne creer_personne();
26
27 /* @requires : nothing
28    @assigns : nothing
29    @ensures : retourne une liste de personne vide */
30 liste_personne creer_liste_personne();
31
32 /* @requires : une liste de personne valide
33    @assigns : nothing
34    @ensures : retourne 1 si la liste est vide et 0 sinon */
35 int liste_personne_vide(liste_personne);
36
37 /* @requires : pointeur sur liste de personne valide et une personne
    valide, non presente dans la liste
38    @assigns : la liste de personne
39    @ensures : la personne a ete inseree en tete de liste */
40 void inserer_liste_personne(liste_personne*, personne);
41
42 /* @requires : pointeur sur liste de personne et personne valides
43    @assigns : la liste de personne *listPer
44    @ensures : si per est presente dans *listPer, retire per de *listPer
        et retourne 0; sinon, retourne -1; */
45 int retirer_liste_personne(liste_personne*, personne);
46
47 #endif /* LISTPER_H */
```

4.4 tas.h

```
1  #ifndef TAS_H
2  #define TAS_H
3
4  #include "listePersonne.h"
5
6  typedef struct tas_base {
7      personne* tableau;
8      int prochain_vide;
9  } * tas;
10
11  /* @requires : nothing
12   @assigns : nothing
13   @ensures : retourne un tas vide */
14  tas creer_tas();
15
16  /* @requires : tas valide
17   @assigns : nothing
18   @ensures : retourne 1 si tas vide, 0 sinon */
19  int tas_vide(tas);
20
21  /* @requires : pointeur sur tas valide et personne valide, non presente
22   dans le tas
23   @assigns : le tas
24   @ensures : personne inseree a la bonne place dans le tas */
25  void inserer_tas(tas*, personne);
26
27  /* @requires : pointeur sur tas valide
28   @assigns : le tas
29   @ensures : retirer personne en conservant nature de tas */
30  personne retirer_tas(tas*);
31
32  /* @requires : pointeur sur tas valide et indice valide
33   @assigns : nothing
34   @ensures : renvoi de la priorite de la personne concernee */
35  int personne_priorite(tas*, int);
36
37  /* @requires : pointeur sur tas valide
38   @assigns : le tas
39   @ensures : echange la racine avec le fils droit */
40  void echanger_racine_fd(tas*, int i);
41
42  /* @requires : pointeur sur tas valide
43   @assigns : le tas
44   @ensures : echange la racine avec le fils gauche */
45  void echanger_racine_fg(tas* t, int i);
46
47  /* @requires : pointeur sur tas valide
48   @assigns : le tas
49   @ensures : reequilibrer le tas apres retrait d'un element et conserver
50   sa structure */
51  void reorganiser_tas_retirer(tas*);
52
53  /* @requires : indice valide
54   @assigns : nothing
55   @ensures : la parite de ind (retourne 1 si impair, 0 sinon) */
56  int est_pair(int ind);
```

```

55
56  /* @requires : pointeur sur tas valide
57     @assigns : le tas
58     @ensures : reequilibrer le tas apres ajout d'un element et conserver
        sa structure */
59  void reorganiser_tas_inserer(tas*);
60
61  /* @requires : pointeur sur tas valide
62     @assigns : le tas
63     @ensures : liberer la zone memoire associee au tas */
64  void liberer_tas(tas*);
65
66  #endif /* TAS_H */

```

5 Choix du langage de programmation

Le langage utilisé pour le projet sera le C. Le projet sera découpé selon les différents modules évoqués plus haut. Néanmoins, il serait possible selon l'avancement du projet de créer une interface utilisateur dans une application web s'appuyant sur une base de données qui permettra à un utilisateur de modifier, supprimer, ajouter des choix valides à l'aide de la structure de base de données. Le programme en lui-même pourrait quant à lui récupérer les classes de cette base de données pour les traiter.