```
# Ceci est un commentaire

# Note préalable - j'ai ici été face à deux interprétations possibles de la consigne de l'exercice :

# (1) développer un algorithme permettant d'assigner une liste de participants à une liste de stands existants en
# fonction de préférences préalablement renseignées
# le programme s'y exécute en une seule boucle et en une seule exécution à partir des données connues

# (2) développer une architecture de plusieurs programmes qui régissent l'ensemble de la journée de convention
# prenant ainsi en compte les arrivées et départs de chaque participant à la convention et sur chaque stand

# j'ai choisi de partir ici sur l'interprétation (1) car :
# - (2) demande de garder en mémoire tous les choix pris au cours de la journée et de considérer le temps passé sur
# chaque stand comme une donnée invariante
# - (2) demande de réfléchir à une configuration matérielle, permettant à chaque participant de notifier le système
# central de ses allées et venues
# - (2) conduirait en l'état à une expérience utilisateur non viable car :
#               - on ne prend pas en compte les temps de déplacements
#               - les participants ne peuvent pas refuser les choix proposés, ils ne peuvent pas non plus faire une pause ou
# partir plus tôt

# Dans les faits le choix (2) serait certainement celui envisagé par le client mais ne pouvant arriver à un tel résultat dans
# le temps imparti je choisis de suivre le scénario (1)
# Dans les faits il me serait donc nécessaire de discuter en amont du cahier des charges et discuter avec lui de ces
# potentiels problèmes d'utilisabilité

############################################################

# Partant donc sur le scénario (1), je dois considérer que les listes de participants et de stands sont déjà connues
# et que les participants sont déjà tous sur place lors de l'exécution du programme.


# la classe Convention référence tous les objets Stand, classés par numéro de stand allant de 0 à X
# ainsi que tous les Attendee (les participants)
# elle contient les méthodes principales du programme

BEGIN

class Convention

        private Stand[] standArray
        private Attendee[] attendeeArray

        bool dataIsLoaded

        # le constructeur charge la liste des stands, ainsi que la liste ordonnée des participants
        # avec leurs préférences respectives

        public Convention(string standArrayFileName, string attendeeArrayFileName)

                if(FileExists(standArrayFileName) && FileExists(attendeeArrayFileName))
                        standArray = LoadFileAsStandArray(standArrayFileName)
                        attendeeArray = SortAttendees(LoadFileAsAttendeeArray(attendeeArrayFileName))

                        dataIsLoaded = true
                else dataIsLoaded = false


        # parcoure la liste des stands préférés de l'attendee et sélectionne le premier qui a des places disponibles
        # si un stand est trouvé alors on y réserve une place et on y assigne le participant
        public AssignAttendeesToStands()

                bool vacantStandFound

                # parcourt l'ensemble de la liste des participants en commençant par le premier arrivé
                foreach(Attendee currentAttendee in attendeeArray)

                        vacantStandFound = False

                        # parcourt l'ensemble des préférences d'affectations du participant
                        for(int i = 0, i< currentAttendee.standIDPreferences.lenght, i++)
```

```
                                        Stand currentStand = currentAttendee.standIDPreferences[i]
                                        # si le stand a des places disponibles on y affecte le participant
                                        # et on le retire de sa liste de préférences
                                        if(currentStand.IsStandAvailable())
                                                currentAttendee.targettedStand = currentStand.standID
                                                currentStand.AddAttendeeToStand(currentAttendee)
                                                attendee.standIDPreferences.removeElementInList(i)
                                                vacantStandFound = True
                                                print("Attendee affected to stand"+currentStand.standID)

                                        # si un stand a été trouvé on sort de la boucle fort
                                        if(vacantStandFound) break

                                # si aucun stand libre n'a été trouvé, on sélectionne le premier choix
                                # de l'attendee et on le met sur la liste d'attente
                                if(not vacantStandFound)
                                        if(currentAttendee.standIDPreferences > 0)
                                                Stand currentStand = currentAttendee.standIDPreferences[0]
                                                currentAttendee.targettedStand = currentStand.standID
                                standArray[attendee.standIDPreferences[0]].AddAttendeeToQueue(attendee)
                                                print("Attendee on waiting queue at stand"+currentStand.standID)

                                        else print("No stand preference found for attendee "+attendee.ID)

                        print("Affectation done !")


        public Attendee[] SortAttendees()
                ## ici une méthode pour trier le tableau de participants en fonction de leur ordre d'arrivée (arrivalRank)

        public Stand[] LoadFileAsStandArray(string standArrayFileName)
                ## ici une méthode pour charger un fichier et renvoyer un array de Stand

        public Attendee[] LoadFileAsAttendeeArray(string attendeeArrayFileName)
                ## ici une méthode pour charger un fichier et renvoyer un array d'Attendee



# cette classe contient les informations permettant l'affectation du participant
class Attendee
        # l'ID sert en cas de stand complet, et s'ajoute à sa file d'attente
        public int ID

        # stocke un entier correspondant à l'ordre d'arrivée à la convention
        public int arrivalRank

        # stocke la liste des IDs des stands dans l'ordre de préférence, sa taille est dynamique et permet de retirer ses préférences
        # tout au long de son parcours
        public list:int standIDPreferencesList

        # stocke l'ID du stand choisi
        public int targettedStand

        public Attendee()
                standIDPreferencesList = new list:int


# la classe Stand contient un ID, des places restantes
class Stand
        public int standID
        private int placesRemaining
        private list:int standAttendees
        private list:int waitingQueue

        # retourne vrai si le stand a des places disponibles, faux sinon
        public bool IsStandAvailable()
                if (placesRemaining > 0) return true
                else return false


        public void AddAttendeeToStand(Attendee attendee)
                standAttendees.Add(attendee.ID)
```

```
                placesRemaining -=1

        public void AddAttendeeToQueue(Attendee attendee)
                waitingQueue.Add(attendee.ID)

        public Stand(int id, int p)
                standID = id
                placesRemaining = p
                standAttendees = new list:int
                waitingQueue = new list:int


###
# On exécute ici notre programme
# on considère les noms "standFile.json" "attendeeFile.json" comme étant les fichiers donnés par la convention avec la liste des
# stands et des participants
###

Convention convention = new Convention("standFile.json", "attendeeFile.json")

if(convention.dataIsLoaded) convention.AssignAttendeesToStands()

else print("Data Not Found")

END


## Optimisations possibles :

# - le temps moyen passé par stand n'a ici pas été utilisé. Dans le cas où tous les stands sont pleins, il conviendrait de spécifier un
# temps limite où l'attente est considérée comme soutenable. Dans ce cas on rajouterait dans la class Stand la donnée de temps
# moyen passé sur le-dit stand, et on mesurerait par exemple parmi les 10 premiers choix le stand où le temps d'attente sera le plus f
# faible

# - si on oriente le développement vers le scénario (2) décrit plus haut, de nombreuses optimisations sont envisageables :

#       - on peut calculer le parcours d'un utilisateur en fonction des parcours calculés pour les utilisateurs arrivés plus tôt,
#       en estimant par exemple les moments où un stand sera supposé libre

#       - on peut envisager de voir les parcours comme dynamiques, si une place se libère, on peut réorienter
#       les priorités des participants les plus proches

#       - on peut également prendre en compte la quantité de déplacement accumulée sur la journée et préférer un stand
#       moins bien classé mais plus logique au regard du parcours total
```