(19-20) **Algorithmique et structures de données**

Corrigé de la série n°4

Ex1 :

```
typecellule  * instete(typecellule *tete, int nb)
{typecellule *nouv ;
 nouv = (typecellule *)malloc(sizeof(typecellule)) ;
nouv->inf = nb ;
nouv->suiv = tete ;
return nouv ;
}
```

Ex2 :

```
typecellule * constrlistealenvers()
{int nb, i ;
 typecellule *tete = NULL ;
 for (i=1 ; i <=10 ; i++)
    {scanf("%d", &nb) ;
     tete = instete(tete, nb);
     }
  return tete;
 }
```

Ex3 :

```
void affichage(typecellule * tete)
{typecellule * p ;
 p = tete ;
 while (p != NULL)
     {printf("%d ", p->inf);
       p = p ->suiv;
      }
}
```

Ex4 :

```
typecellule *insqueue(typecellule * tete, int nb)
{typecellule *nouv, *p ;
 nouv = (typecellule *)malloc(sizeof(typecellule)) ;
 nouv->inf = nb ;
 nouv->suiv = NULL ;
```

```
 if (tete == NULL)
   tete =  nouv ;
 else
    {for (p=tete;p->suiv!=NULL ;p=p->sui)
         {}
     p->suiv = nouv;
    }
   return tete;
 }
```

Ex5 :

```
typecellule * constrlistealendroit()
{int nb, i ;
 typecellule *tete = NULL ;
 for (i=1 ; i <=10 ; i++)
    {scanf("%d", &nb) ;
     tete = insqueue(tete, nb);
     }
  return tete;
 }
```

Ex6 :

```
Dliste *insqueue(Dliste *list, int inf)
{
   if (list != NULL) /* On vérifie si notre liste a été allouée */
   {
      struct cellule *newl = malloc(sizeof *newl); /* Création d'une
                        nouvelle cellule */
      if (newl != NULL) /* On vérifie si le malloc n'a pas échoué */
      {
         newl->inf = inf; /* On 'enregistre' notre donnée */
         newl->suiv = NULL; /* On fait pointer suiv vers NULL */
         if (list->queue == NULL) /* Cas où notre liste est vide
                                     (pointeur vers fin de liste à  NULL) */
         {
            newl->prec = NULL; /* On fait pointer prec vers NULL */
            list->tete = newl; /* On fait pointer la tête de liste vers
                                  le nouvel élément */
            list->queue = newl; /* On fait pointer la fin de liste vers
                                   le nouvel élément */
          }
```

```
        else /* Cas où des éléments sont déjà présents dans notre liste
              */
         {
           list->queue->suiv = newl; /* On relie le dernier élément de
                                        la liste vers notre nouvel élément */
           newl->prec = list->queue; /* On fait pointer prec vers le dernier
                                        élément de la liste */
           list->queue = newl; /* On fait pointer la fin de liste vers notre
                                  nouvel élément  */
          }
        list->long++; /* Incrémentation de la taille de la liste */
     }
   }
   return list; /* on retourne notre nouvelle liste */
}
```

Ex7 :

```
Dlist *instete(Dlist *list, int inf)
{
   if (list != NULL)
   {
      struct node *new = malloc(sizeof *new);
      if (newl != NULL)
      {
         newl->inf = inf;
         newl->prec = NULL;
         if (list->queue == NULL)
         {
            newl->suiv = NULL;
            list->tete = newl;
            list->queue = newl;
         }
         else
         {
            list->tete->prec = newl;
            newl->suiv = list->tete;
            list->tete = newl;
         }
         list->long++;
     }
   }
   return list;}
```

Ex8 :

```
Dliste *list_insert(Dliste *list, int inf, int position)
{
   if (list != NULL)
   {
      struct cellule *temp = list->tete;
      int i = 1;
      while (temp != NULL && i <= position)
      {
         if (position == i)
         {
            if (temp->suiv == NULL)
            {
               list = insqueue(list, inf);
            }
            else if (temp->prec == NULL)
            {
               list = instete(p_list, data);
            }
            else
            {
               struct cellule *newl = malloc(sizeof *new);;
               if (newl != NULL)
               {
                  newl->inf = inf;

                  newl->prec = temp->prec;
                  newl->suiv = temp;
                  temp->prec->suiv = newl ;
                  temp->prec = newl ;

                  list->long++;


               }
            }
         }
         else
         {
            temp = temp->suiv;
          }
```

```
            i++;
        }
    }
    return list;
}
```