Algorithmique — IN102
TD 2

9 décembre 2005

```
typedef struct cell {
  int val;
  struct cell *suiv;
} cellule;
typedef cellule * liste;
```

Exercice 1

```
liste cons(int v, liste vs) {
  liste nouv = new(cellule);
  nouv→val = v; nouv→suiv = vs;
  return nouv;
}
```

Exercice 2

```
void printListeIter( liste vs) {
  liste p = vs;
  while (p ≠ NULL) { cout « p→val « "␣"; p = p→suiv; }
  cout « endl;
}

void printListeRec( liste vs) {
  if (vs == NULL)
    cout « endl;
  else {
    cout « vs→val « "␣";
    printListeRec(queue(vs));
  }
}
```

Exercice 3 *Avec recopie et sans récupération de la mémoire :*

```
liste insere(int v, liste vs) {
  if (vs == NULL)
    return cons(v, NULL);
   else
     if (vs→val < v)
       return cons(vs→val, insere(v, vs→suiv));
     else
       return cons(v, vs);
}
```

Exercice 4 *Sans récupération mémoire :*

```
liste triInsertion ( liste vs) {
  liste triee = NULL;
  while (vs ≠ NULL) {
    triee = insere(vs→val, triee);
    vs = vs→suiv;
  }
  return triee;
}
```

Exercice 5 *On utilise une fonction d'inversion qui va faciliter la construction du résultat.*

```
liste inversion( liste vs, liste accu) {
  if (vs == NULL)
    return accu;
  else
    return
      inversion(queue(vs), cons(tete(vs), accu));
}

#define N 100

liste triPaquet(int k, liste vs) {
  liste *tab = new liste[k];
  liste p = vs;

  for (int i = 0; i < k; i++)
    tab[i] = NULL;

  while (p ≠ NULL) {
    tab[(p→val*k)/N] = insere(p→val, tab[(p→val*k)/N]);
    p = p→suiv;
  }

  liste res = NULL;

  for(int i = 0; i < k; i++) {
    p = tab[i];
    while (p ≠ NULL) {
      res = cons(p→val,res);
      p = p→suiv;
    }
  }
  return inversion(res, NULL);
}
```

Écrire une fonction libérant l'ensemble des cellules occuppées par une liste chaînée.

Exercice 6

```
void libere( liste vs) {
  if (vs ≠ NULL) {
    libere(vs→suiv); // attention à ne pas
```

```
    delete [] vs;      // inverser ces deux instructions !!
  }
}
```

Exercice 7

```
void insere2(int v, liste *pl) {
  // NB: la liste est passée par référence
  if (*pl == NULL)
    *pl = cons(v, *pl);
  else {
     liste pc = *pl;
     liste prec = NULL; // ptr vers la cellule précédente
    while ((pc ≠ NULL) && (pc→val < v)) {
      prec = pc; pc = pc→suiv;
    }
    if (prec == NULL) // On n'a pas avancé
      *pl = cons(v, pc);
    else
      prec→suiv = cons(v, prec→suiv);
  }
}
```

Exercice 8

```
liste triPaquet2(int k, liste vs) {
  liste *tab = new liste[k];
  liste p = vs;

  for (int i = 0; i < k; i++)
    tab[i] = NULL;

  while (p ≠ NULL) {
    insere2(p→val, &tab[(p→val*k)/N]);
    p = p→suiv;
  }

  liste res = tab[0];

  for(int i = 1; i < k; i++) {
    p = tab[i];
    if (p ≠ NULL) {
      while (p→suiv ≠ NULL)
  p = p→suiv;
      p→suiv = res;
      res = tab[i];
    }
  }
  return res;
}
```

Exercice 9

```
1.  file  fileVide () {
      file  f = new(struct strfile);
      f→pre = f→der = NULL;
      return f;
    }

    bool estVide( file  f) {
      return f→pre == NULL;
    }


2.  void arrive(int n,  file  f) {
      liste  nouv = new(cellule);
      nouv→val=n; nouv→suiv = NULL;
      if (estVide(f))
        f→pre = f→der = nouv;
      else {
        f→der→suiv = nouv;
        f→der = f→der→suiv;
      }
    }

    int suivant( file  f) {
      if (estVide(f)) erreur("suivant(File Vide)");
      int s = f→pre→val;
      if (f→pre == f→der)
        f→der = NULL;
      f→pre = f→pre→suiv;
      return s;
    }
```