TD/TP de Programmation – Correction de la Planche 3
Trieuses d'entiers et trieuses génériques

1 Trieuse d'entiers `IntSelectionSorter`

1.1 Initialisation et affichage de la trace

```
void
IntSelectionSorter_Init (IntSelectionSorter * s, int values [], int nb_values)
{
  s->values= values;
  s->nb_values  = nb_values;
  s->nb_unsorted= nb_values;
  s->selected_index= -1;
}
```

```
void
IntSelectionSorter_Print (IntSelectionSorter const * s, FILE * file)
{
  for (int k= 0; k < s->nb_values; k++) {
    char star= (k == s->selected_index ) ? '*' : '␣';
    char bar = (k == s->nb_unsorted - 1) ? '|' : '␣';
    fprintf (file, "%c%d%c%c", star,  s->values [k], star, bar);
  }
  fprintf (file, "\n");
}
```

```
int
MainTest_IntSelectionSorter_1 (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  IntSelectionSorter sorter;
  IntSelectionSorter_Init (& sorter, values, 7);
  IntSelectionSorter_Print (& sorter, stdout);
  return 0;
}
```

1.2 Sélection de l'élément maximal

```
void
IntSelectionSorter_SelectMax (IntSelectionSorter * s)
{
  int max_k= 0;
  for (int k= 1; k < s->nb_unsorted; k++)
    if (s->values [k] > s->values [max_k])
      max_k= k;
  s->selected_index= max_k;
}
```

```
int
MainTest_IntSelectionSorter_2 (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  IntSelectionSorter sorter;
  IntSelectionSorter_Init (& sorter, values, 7);
  IntSelectionSorter_SelectMax (& sorter);
  IntSelectionSorter_Print (& sorter, stdout);
  return 0;
}
```

1.3 Échange de deux éléments

```
void
IntSelectionSorter_SwapValues (IntSelectionSorter * s, int k1, int k2)
{
  int value1    = s->values [k1];
  s->values [k1]= s->values [k2];
  s->values [k2]= value1;
}
```

```
int
MainTest_IntSelectionSorter_3 (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  IntSelectionSorter sorter;
  IntSelectionSorter_Init  (& sorter, values, 7);
  IntSelectionSorter_Print (& sorter, stdout);
  IntSelectionSorter_SwapValues (& sorter, 5, 6);
  IntSelectionSorter_Print (& sorter, stdout);
  return 0;
}
```

1.4 Étape élémentaire du tri : la sélection suivie de l'échange

```
void
IntSelectionSorter_DoSortStep (IntSelectionSorter * s, FILE * trace_file)
{
  IntSelectionSorter_SelectMax (s);
  if (trace_file != NULL) IntSelectionSorter_Print(s, trace_file);
  IntSelectionSorter_SwapValues (s, s->selected_index, s->nb_unsorted-1);
  s->nb_unsorted--;
  s->selected_index= -1;
}
```

```
int
MainTest_IntSelectionSorter_4 (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  IntSelectionSorter sorter;
  IntSelectionSorter_Init       (& sorter, values, 7);
  IntSelectionSorter_DoSortStep (& sorter, stdout);
  IntSelectionSorter_Print      (& sorter, stdout);
  return 0;
}
```

1.5 Tri sélection complet

```
void
IntSelectionSorter_Sort (IntSelectionSorter * s, FILE * trace_file)
{
  while (s->nb_unsorted > 1)
    IntSelectionSorter_DoSortStep (s, trace_file);
  IntSelectionSorter_Print (s, stdout);
}
```

```
int
MainTest_IntSelectionSorter_5 (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  IntSelectionSorter sorter;
  IntSelectionSorter_Init  (& sorter, values, 7);
  IntSelectionSorter_Sort  (& sorter, stdout);
  return 0;
}
```

2 Trieuse d'entiers `IntInsertionSorter`

2.1 Initialisation et affichage de la trace

```
void
IntInsertionSorter_Init (IntInsertionSorter * s, int values [], int nb_values)
{
  s->values= values;
  s->nb_values= nb_values;
  s->nb_sorted= nb_values ? 1 : 0;
  s->insertion_index= -1;
}
```

```
void
IntInsertionSorter_Print (IntInsertionSorter const * s, FILE * file)
{
  for (int k= 0; k < s->nb_values; k++) {
    char star= (k == s->insertion_index) ? '*' : '␣';
    char bar = (k == s->nb_sorted - 1  ) ? '|' : '␣';
    fprintf (file, "%c%d%c%c", star,  s->values [k], star, bar);
  }
  fprintf (file, "\n");
}
```

```
int MainTest_IntInsertionSorter_1 (void)
{
  int values []=  { 66, 11, 99, 33, 44, 22, 55 };
  IntInsertionSorter sorter;
  IntInsertionSorter_Init  (& sorter, values, 7);
  IntInsertionSorter_Print (& sorter, stdout);
  return 0;
}
```

2.2 Étape élémentaire du tri : l'insertion d'un élément dans la partie triée

```
void
IntInsertionSorter_Insert (IntInsertionSorter * s)
{
  int saved_value= s->values [s->nb_sorted];
  int hole;
  for (hole= s->nb_sorted; hole >= 1; hole--) {
    if (s->values [hole-1] <= saved_value) break;
    s->values [hole]= s->values [hole-1];
  }
  s->values [hole]= saved_value;
  s->insertion_index= hole;
  s->nb_sorted++;
}
```

```
void
IntInsertionSorter_DoSortStep (IntInsertionSorter * s, FILE * trace_file)
{
  IntInsertionSorter_Insert (s);
  if (trace_file != NULL) IntInsertionSorter_Print (s, trace_file);
}
```

```
int MainTest_IntInsertionSorter_2 (void)
{
  int values []=  { 66, 11, 99, 33, 44, 22, 55 };
  IntInsertionSorter sorter;
  IntInsertionSorter_Init       (& sorter, values, 7);
  IntInsertionSorter_DoSortStep (& sorter, stdout);
  return 0;
}
```

2.3 Tri Insertion complet

```
void
IntInsertionSorter_Sort (IntInsertionSorter * s, FILE * trace_file)
{
  if (trace_file != NULL) IntInsertionSorter_Print (s, trace_file);
  while (s->nb_sorted < s->nb_values) {
    IntInsertionSorter_Insert (s);
    if (trace_file != NULL) IntInsertionSorter_Print (s, trace_file);
  }
}
```

```
int
MainTest_IntInsertionSorter_3 (void)
{
  int values []=  { 66, 11, 99, 33, 44, 22, 55 };
  IntInsertionSorter sorter;
  IntInsertionSorter_Init  (& sorter, values, 7);
  IntInsertionSorter_Sort  (& sorter, stdout);
  return 0;
}
```

3 Données génériques : exemples pour les entiers et les chaînes

```
int
CompareAsInt (void const * data1, void const * data2)
{
  int const * integer1= data1;
  int const * integer2= data2;

  if (* integer1 < * integer2) return -1;
  if (* integer1 > * integer2) return +1;
  return 0;
}
```

```
int
CompareAsString (void const * data1, void const * data2)
{
  char * const * str1= data1;
  char * const * str2= data2;
  return strcmp (* str1, * str2);
}
```

```
void
PrintAsInt (void const * data, FILE * file)
{
  int const * integer= data;
  fprintf (file, "%2d", * integer);
}
```

```
void
PrintAsString (void const * data, FILE * file)
{
  char * const * str= data;
  fprintf (file, "%s", * str);
}
```

4 Trieuse générique `SelectionSorter`

4.1 Initialisation et affichage de la trace

```
void
SelectionSorter_Init (SelectionSorter * s, void * cells, int nb_cells,
                      CompareFunc * compare, PrintFunc * print, int cell_size)
{
  s->bytes= cells;
  s->nb_cells= nb_cells;
  s->nb_unsorted= nb_cells;
  s->selected_index= -1;

  s->compare= compare;
  s->print= print;
  s->cell_size= cell_size;
}
```

```
void *
SelectionSorter_CellAt (SelectionSorter const * s, int index)
{
  return  s->bytes + index * s->cell_size;
}
```

```
void
SelectionSorter_Print (SelectionSorter const * s, FILE * file)
{
  for (int k= 0; k < s->nb_cells; k++) {
    void * data= SelectionSorter_CellAt (s, k);
    char star= (k == s->selected_index ) ? '*' : '␣';
    char bar = (k == s->nb_unsorted - 1) ? '|' : '␣';
    fprintf (file, "%c", star);
    s->print (data, file);
    fprintf (file, "%c%c",  star, bar);
  }
  fprintf (file, "\n");
}
```

```
int
MainTest_SelectionSorter_1a (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  SelectionSorter sorter;
  SelectionSorter_Init (& sorter, values, 7,
                        CompareAsInt, PrintAsInt,
                        sizeof * values);
  SelectionSorter_Print (& sorter, stdout);
  return 0;
}
```

```
int
MainTest_SelectionSorter_1b (void)
{
  char * values []= { "END", "COW", "BAR", "FLU", "GAL", "ICE", "ART" };
  SelectionSorter sorter;
  SelectionSorter_Init (& sorter, values, 7,
                        CompareAsString, PrintAsString,
                        sizeof * values);
  SelectionSorter_Print (& sorter, stdout);
  return 0;
}
```

4.2 Sélection de l'élément maximal

```
void
SelectionSorter_SelectMax (SelectionSorter * s)
{
  int max_index= 0;
  for (int index= 1; index < s->nb_unsorted; index++) {
    void * data     = SelectionSorter_CellAt (s, index);
    void * max_data= SelectionSorter_CellAt (s, max_index);
    if (s->compare (data, max_data) > 0) max_index= index;
  }
  s->selected_index= max_index;
}
```

```
int
MainTest_SelectionSorter_2a (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  SelectionSorter sorter;
  SelectionSorter_Init (& sorter, values, 7,
                        CompareAsInt, PrintAsInt,
                        sizeof * values);
  SelectionSorter_SelectMax (& sorter);
  SelectionSorter_Print (& sorter, stdout);
  return 0;
}
```

```
int
MainTest_SelectionSorter_2b (void)
{
  char * values []= { "END", "COW", "BAR", "FLU", "GAL", "ICE", "ART" };
  SelectionSorter sorter;
  SelectionSorter_Init (& sorter, values, 7,
                        CompareAsString, PrintAsString,
                        sizeof * values);
  SelectionSorter_SelectMax (& sorter);
  SelectionSorter_Print (& sorter, stdout);
  return 0;
}
```

4.3 Échange de deux éléments

```
void
ArrayOfBytes_Swap (char bytes1 [], char bytes2 [], int nb_bytes)
{
  for (int k= 0; k < nb_bytes; k++) {
    char saved1= bytes1 [k];
    bytes1 [k]= bytes2 [k];
    bytes2 [k]= saved1;
  }
}
```

```
void
SelectionSorter_SwapCellsAtIndex (SelectionSorter * s, int k1, int k2)
{
  void * data1= SelectionSorter_CellAt (s, k1);
  void * data2= SelectionSorter_CellAt (s, k2);
  ArrayOfBytes_Swap (data1, data2, s->cell_size);
}
```

```
int MainTest_SelectionSorter_3a (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  SelectionSorter sorter;
  SelectionSorter_Init (& sorter, values, 7,
                        CompareAsInt, PrintAsInt,
                        sizeof * values);
  SelectionSorter_Print (& sorter, stdout);
  SelectionSorter_SwapCellsAtIndex (& sorter, 5, 6);
  SelectionSorter_Print (& sorter, stdout);
  return 0;
}
```

```
int MainTest_SelectionSorter_3b (void)
{
  char * values []= { "END", "COW", "BAR", "FLU", "GAL", "ICE", "ART" };
  SelectionSorter sorter;
  SelectionSorter_Init (& sorter, values, 7,
                        CompareAsString, PrintAsString,
                        sizeof * values);
  SelectionSorter_Print (& sorter, stdout);
  SelectionSorter_SwapCellsAtIndex (& sorter, 5, 6);
  SelectionSorter_Print (& sorter, stdout);
  return 0;
}
```

4.4 Étape élémentaire du tri : la sélection suivie de l'échange

```
void
SelectionSorter_DoSortStep (SelectionSorter * s, FILE * trace_file)
{
  SelectionSorter_SelectMax (s);
  if (trace_file != NULL) SelectionSorter_Print(s, trace_file);
  SelectionSorter_SwapCellsAtIndex (s, s->selected_index, s->nb_unsorted-1);
  s->nb_unsorted--;
  s->selected_index= -1;
  if (trace_file != NULL) SelectionSorter_Print(s, trace_file);
}
```

```
int MainTest_SelectionSorter_4a (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  SelectionSorter sorter;
  SelectionSorter_Init (& sorter, values, 7,
                        CompareAsInt, PrintAsInt,
                        sizeof * values);
  SelectionSorter_DoSortStep (& sorter, stdout);
  SelectionSorter_DoSortStep (& sorter, stdout);
  SelectionSorter_DoSortStep (& sorter, stdout);
  return 0;
}
```

```
int MainTest_SelectionSorter_4b (void)
{
  char * values []= { "END", "COW", "BAR", "FLU", "GAL", "ICE", "ART" };
  SelectionSorter sorter;
  SelectionSorter_Init (& sorter, values, 7,
                        CompareAsString, PrintAsString,
                        sizeof * values);
  SelectionSorter_DoSortStep (& sorter, stdout);
  SelectionSorter_DoSortStep (& sorter, stdout);
  SelectionSorter_DoSortStep (& sorter, stdout);
  return 0;
}
```

4.5 Tri Sélection complet

```
void
SelectionSorter_Sort (SelectionSorter * s, FILE * trace_file)
{
  while (s->nb_unsorted > 1) {
    SelectionSorter_DoSortStep (s, trace_file);
  }
}
```

```
int
MainTest_SelectionSorter_5a (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  SelectionSorter sorter;
  SelectionSorter_Init  (& sorter, values, 7,
                         CompareAsInt, PrintAsInt,
                         sizeof * values);
  SelectionSorter_Sort  (& sorter, stdout);
  return 0;
}
```

```
int
MainTest_SelectionSorter_5b (void)
{
  char * values []= { "END", "COW", "BAR", "FLU", "GAL", "ICE", "ART" };
  SelectionSorter sorter;
  SelectionSorter_Init  (& sorter, values, 7,
                         CompareAsString, PrintAsString,
                         sizeof * values);
  SelectionSorter_Sort  (& sorter, stdout);
  return 0;
}
```

5 Trieuse générique `InsertionSorter`

5.1 Initialisation et affichage de la trace

```
void
InsertionSorter_Init (InsertionSorter * s, void * cells, int nb_cells,
                      CompareFunc * compare, PrintFunc * print, int cell_size)
{
  s->bytes= cells;
  s->nb_cells= nb_cells;
  s->nb_sorted= nb_cells ? 1 : 0;
  s->insertion_index= -1;

  s->compare= compare;
  s->print= print;
  s->cell_size= cell_size;
}
```

```
void *
InsertionSorter_CellAt (InsertionSorter const * s, int index)
{
  return  s->bytes + index * s->cell_size;
}
```

```
void
InsertionSorter_Print (InsertionSorter const * s, FILE * file)
{
  for (int k= 0; k < s->nb_cells; k++) {
    void * data= InsertionSorter_CellAt (s, k);
    char star= (k == s->insertion_index ) ? '*' : '␣';
    char bar = (k == s->nb_sorted - 1) ? '|' : '␣';
    fprintf (file, "%c", star);
    s->print (data, file);
    fprintf (file, "%c%c",  star, bar);
  }
  fprintf (file, "\n");
}
```

```
int
MainTest_InsertionSorter_1a (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  InsertionSorter sorter;
  InsertionSorter_Init (& sorter, values, 7,
                        CompareAsInt, PrintAsInt,
                        sizeof * values);
  InsertionSorter_Print (& sorter, stdout);
  return 0;
}
```

```
int
MainTest_InsertionSorter_1b (void)
{
  char * values []= { "END", "COW", "BAR", "FLU", "GAL", "ICE", "ART" };
  InsertionSorter sorter;
  InsertionSorter_Init (& sorter, values, 7,
                        CompareAsString, PrintAsString,
                        sizeof * values);
  InsertionSorter_Print (& sorter, stdout);
  return 0;
}
```

5.2 Étape élémentaire du tri : l'insertion d'un élément dans la partie triée

```
void
ArrayOfBytes_CopyFrom (char dest [], char source [], int nb_bytes)
{
  for (int k= 0; k < nb_bytes; k++) {
    dest [k]= source [k];
  }
}
```

```
#define CELL_CAPACITY 1024

void
InsertionSorter_Insert (InsertionSorter * s)
{
  char saved [CELL_CAPACITY];
  void * to_be_inserted= InsertionSorter_CellAt (s, s->nb_sorted);
  ArrayOfBytes_CopyFrom (saved, to_be_inserted, s->cell_size);
  int hole;
  for (hole= s->nb_sorted; hole >= 1; hole--) {
    void * curr= InsertionSorter_CellAt (s, hole);
    void * prev= InsertionSorter_CellAt (s, hole-1);
    if (s->compare (prev, saved) <= 0) break;
    ArrayOfBytes_CopyFrom (curr, prev, s->cell_size);
  }
  void * curr= InsertionSorter_CellAt (s, hole);
  ArrayOfBytes_CopyFrom (curr, saved, s->cell_size);
  s->insertion_index= hole;
  s->nb_sorted++;
}
```

```
void
InsertionSorter_DoSortStep (InsertionSorter * s, FILE * trace_file)
{
  InsertionSorter_Insert (s);
  if (trace_file != NULL) InsertionSorter_Print (s, trace_file);
}
```

```
int
MainTest_InsertionSorter_2a (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  InsertionSorter sorter;
  InsertionSorter_Init (& sorter, values, 7,
                        CompareAsInt, PrintAsInt,
                        sizeof * values);
  InsertionSorter_Print (& sorter, stdout);
  InsertionSorter_DoSortStep (& sorter, stdout);
  return 0;
}
```

```
int
MainTest_InsertionSorter_2b (void)
{
  char * values []= { "END", "COW", "BAR", "FLU", "GAL", "ICE", "ART" };
  InsertionSorter sorter;
  InsertionSorter_Init (& sorter, values, 7,
                        CompareAsString, PrintAsString,
                        sizeof * values);
  InsertionSorter_Print (& sorter, stdout);
  InsertionSorter_DoSortStep (& sorter, stdout);
  return 0;
}
```

5.3 Tri Insertion complet

```
void
InsertionSorter_Sort (InsertionSorter * s, FILE * trace_file)
{
  if (trace_file != NULL) InsertionSorter_Print (s, trace_file);
  while (s->nb_sorted < s->nb_cells) {
    InsertionSorter_DoSortStep (s, trace_file);
  }
}
```

```
int
MainTest_InsertionSorter_3a (void)
{
  int values []=  { 55, 33, 22, 66, 77, 99, 11 };
  InsertionSorter sorter;
  InsertionSorter_Init  (& sorter, values, 7,
                         CompareAsInt, PrintAsInt,
                         sizeof * values);
  InsertionSorter_Sort  (& sorter, stdout);
  return 0;
}
```

```
int
MainTest_InsertionSorter_3b (void)
{
  char * values []= {  "COW", "BAR",  "ICE", "GAL", "END",  "FLU", "ART" };
  InsertionSorter sorter;
  InsertionSorter_Init  (& sorter, values, 7,
                         CompareAsString, PrintAsString,
                         sizeof * values);
  InsertionSorter_Sort  (& sorter, stdout);
  return 0;
}
```