GA.CPP

```
#include <math.h>
#include <stdlib.h>
#include <memory.h>
#include <stdio.h>
#include <conio.h>


#include "rand.h"
#include "msortidx.h"

#ifndef NULL
#define NULL (0)
#endif

#undef GA_MUTATION_MASK
#undef GA_CLASSICAL_CROSSING_OVER

// #define GA_MUTATION_MASK  3 // 1/4 bit flip probability.
// #define GA_MUTATION_MASK  7 // 1/8 bit flip probability.
// #define GA_MUTATION_MASK 15 // 1/16 bit flip probability.
// #define GA_MUTATION_MASK 31 // 1/32 bit flip probability.
#define GA_MUTATION_MASK 63 // 1/64 bit flip probability.
// #define GA_MUTATION_MASK 255 // 1/256 bit flip probability.

// #define GA_CLASSICAL_CROSSING_OVER // For canonical testing.

#define GA_ALLOW_DEATH_OF_LEADING

// *******************************************************
// ************* Single organism class *******************
// *******************************************************


class GA_ORGANISM_CLASS
{
  public:

  bool *member_is_on;
  static __int64 *member_powers;
  static int member_n;
  static int member_ref_count;
  static __int64 member_crux_number;
  static RAND_RANDOM_CLASS member_rand_class;

  GA_ORGANISM_CLASS(void);
  ~GA_ORGANISM_CLASS();

  void function_free(void);
  bool function_init(int i_n);

  void function_cross(GA_ORGANISM_CLASS &c_a);
  void function_mutate(void);

  bool operator !=(GA_ORGANISM_CLASS &neq_a);
```

```
  bool operator ==(GA_ORGANISM_CLASS &eq_a);
  GA_ORGANISM_CLASS &operator =(GA_ORGANISM_CLASS &as_a);

  double function_target(bool t_is_reducible);

  void function_print_console(void);
};

__int64 *GA_ORGANISM_CLASS::member_powers = NULL;
int GA_ORGANISM_CLASS::member_n = 0;
int GA_ORGANISM_CLASS::member_ref_count = 0;
__int64 GA_ORGANISM_CLASS::member_crux_number = 0;
RAND_RANDOM_CLASS GA_ORGANISM_CLASS::member_rand_class;


GA_ORGANISM_CLASS::GA_ORGANISM_CLASS(void)
{
  member_is_on = NULL;
}


GA_ORGANISM_CLASS::~GA_ORGANISM_CLASS()
{
  function_free();
}

void GA_ORGANISM_CLASS::function_free(void)
{
  if (member_is_on)
  {
    delete[] member_is_on;
    member_is_on = NULL;

    if (member_ref_count)
    {
      member_ref_count--;

      if (0==member_ref_count)
      {
        if (member_powers)
        {
          delete[] member_powers;
          member_powers = NULL;
        }

        member_n = 0;
        member_crux_number = 0;
      }
    }
  }
}


bool GA_ORGANISM_CLASS::function_init(int i_n)
{
  int i;
```

```
  function_free();

  if (i_n<=0) return false;
  if (i_n>62) return false;

  if (member_n && i_n != member_n) return false;

  member_is_on = new bool[i_n];

  if (!member_is_on)
  {
    function_free();
    return false;
  }

  member_ref_count++;

  if (1==member_ref_count)
  {
    int i;
    __int64 i_power,i_sigma;

    member_powers = new __int64[i_n];
    if (!member_powers)
    {
      function_free();
      return false;
    }

    member_n = i_n;

    for(i=0,i_power=1,i_sigma=0;i<i_n;i++,i_power<<=1)
    {
      member_powers[i] = i_power;
      if (i&1) i_sigma += i_power;
    }

    member_crux_number = i_sigma;

    if (member_rand_class.function_init(i_n)<=0)
    {
      function_free();
      return false;
    }
  }

  // Random initialization.
  for(i=0;i<member_n;i++)
  {
    if (rand()&1) member_is_on[i] = true;
    else member_is_on[i] = false;
  }

  return true;
}
```

```
#ifdef GA_CLASSICAL_CROSSING_OVER

void GA_ORGANISM_CLASS::function_cross(GA_ORGANISM_CLASS &c_a)
{
  int i,c_mid;

  c_mid = member_rand_class.function_random(member_n);

  if (rand()&1)
  {
    for(i=0;i<=c_mid;i++)
    {
      member_is_on[i] = c_a.member_is_on[i];
    }
  }
  else
  {
    for(i=c_mid;i<member_n;i++)
    {
      member_is_on[i] = c_a.member_is_on[i];
    }
  }
}


#else

void GA_ORGANISM_CLASS::function_cross(GA_ORGANISM_CLASS &c_a)
{
  int i,c_difference_count;

  for(i=0,c_difference_count=0;i<member_n;i++)
  {
    if (member_is_on[i] != c_a.member_is_on[i])
    {
      c_difference_count++;
    }
  }

  if (c_difference_count<=1) return;


  if (rand() & 1 || 2==c_difference_count)
  {
    if (member_rand_class.
        function_select_n_out_of_2n(c_difference_count)<0)
    {
      return; // Error.
    }

    for(i=0,c_difference_count=0;i<member_n;i++)
    {
      if (member_is_on[i] != c_a.member_is_on[i])
      {
        if (member_rand_class.
            member_bool_selection_array[c_difference_count])
        {
```

```
          member_is_on[i] = c_a.member_is_on[i];
        }

        c_difference_count++;
      }
    }
  }
  else
  {
    int c_choose;

    c_choose =
    member_rand_class.function_random(c_difference_count);

    if (member_rand_class.
        function_select_m_out_of_n(c_choose,
                                   c_difference_count)<0)
    {
      return; // Error.
    }


    for(i==0,c_difference_count=0;i<member_n;i++)
    {
      if (member_is_on[i] != c_a.member_is_on[i])
      {
        if (member_rand_class.
            member_bool_selection_array[c_difference_count])
        {
          member_is_on[i] = c_a.member_is_on[i];
        }

        c_difference_count++;
      }
    }
  }
}

#endif

void GA_ORGANISM_CLASS::function_mutate(void)
{
  int i;

  for(i=0;i<member_n;i++)
  {
    if (GA_MUTATION_MASK ==
        (rand() & GA_MUTATION_MASK)
       )
    {
      member_is_on[i] = !member_is_on[i];
    }
  }
}


bool GA_ORGANISM_CLASS::operator !=(GA_ORGANISM_CLASS &neq_a)
```

```
{
  int i;

  if (!neq_a.member_is_on && member_is_on) return true;
  if (neq_a.member_is_on && !member_is_on) return true;
  if (!neq_a.member_is_on && !member_is_on) return false;

  for(i=0;i<member_n;i++)
  {
    if (neq_a.member_is_on[i]!=member_is_on[i]) return true;
  }

  return false;
}


bool GA_ORGANISM_CLASS::operator ==(GA_ORGANISM_CLASS &eq_a)
{
  int i;

  if (!eq_a.member_is_on && member_is_on) return false;
  if (eq_a.member_is_on && !member_is_on) return false;
  if (!eq_a.member_is_on && !member_is_on) return true;

  for(i=0;i<member_n;i++)
  {
    if (eq_a.member_is_on[i]!=member_is_on[i]) return false;
  }

  return true;
}


GA_ORGANISM_CLASS &GA_ORGANISM_CLASS::
                   operator =(GA_ORGANISM_CLASS &as_a)
{
  if (!as_a.member_is_on)
  {
    function_free();
    return *this;
  }

  if (!member_is_on)
  {
    if (!function_init(member_n)) return *this;
  }

  memcpy(member_is_on,
         as_a.member_is_on,
         sizeof(bool)*member_n);

  return *this;
}


double GA_ORGANISM_CLASS::
       function_target(bool t_is_reducible)
```

```
{
  int i;
  __int64 t_sum = 0;
  double t_result;

  if (t_is_reducible)
  {
    for(i=0;i<member_n;i++)
    {
      if (member_is_on[i])
      {
        t_sum += member_powers[i];
      }
    }

    t_result = fabs((double)(t_sum - member_crux_number));
  }
  else
  {
    __int64 t_coefficient = 1;

    for(i=0;i<member_n;i++)
    {
      if (member_is_on[i])
      {
        t_sum += member_powers[i];

        if (0==(i&1)) t_coefficient <<= 1;
      }
    }

    // t_coefficient reduces the target (cost) function.
    // By that, the fitness function is highly
    // non linear and jeopardizes the GAs conversion.

    t_result =
    fabs((double)(t_sum - member_crux_number)) /
    t_coefficient;

    // By a second devision by powers of two, powers of 4
    // are obtained.
    t_result /= t_coefficient;

  }

  return t_result;
}


void GA_ORGANISM_CLASS::function_print_console(void)
{
  int i;

  for(i=0;i<member_n;i++)
  {
    if (member_is_on[i]) putchar('1'); else putchar('0');
  }
```

```
}


// ********************************************************
// ********* End of 'Single organism class' ***************
// ********************************************************

// ********************************************************
// ***************** Population class *********************
// ********************************************************

class GA_POPULATION_CLASS
{
  public:

  GA_ORGANISM_CLASS *member_organism_class_array;
  double *member_results_array;

  int member_n_survivors,member_n_population;

  GA_POPULATION_CLASS(void);
  ~GA_POPULATION_CLASS();
  bool function_init(int i_n_survivors);
  void function_free(void);
  void function_select(bool s_is_reducible);
  void function_run(int r_epochs,
                    bool r_is_reducible,
                    int r_n_survivors);
};


GA_POPULATION_CLASS::GA_POPULATION_CLASS(void)
{
  member_organism_class_array = NULL;
  member_results_array = NULL;
  member_n_survivors = 0;
  member_n_population = 0;
}


GA_POPULATION_CLASS::~GA_POPULATION_CLASS()
{
  function_free();
}


bool GA_POPULATION_CLASS::function_init(int i_n_survivors)
{
  int i,i_n_total,i_n_population;

  function_free();

  i_n_population = i_n_survivors << 2;

  i_n_total = i_n_population + i_n_survivors;

  #ifdef GA_ALLOW_DEATH_OF_LEADING
```

```
  i_n_total += i_n_survivors;

  #endif

  member_organism_class_array = new GA_ORGANISM_CLASS[i_n_total];

  if (!member_organism_class_array) return false;

  member_results_array = new double[i_n_total];

  if (!member_results_array)
  {
    function_free();
    return false;
  }

  for(i=0;i<i_n_total;i++)
  {
    if (!member_organism_class_array[i].function_init(52))
    {
      function_free();
      return false;
    }
  }

  member_n_survivors = i_n_survivors;
  member_n_population = i_n_population;

  return true;
}


void GA_POPULATION_CLASS::function_free(void)
{
  if (member_results_array)
  {
    delete[] member_results_array;
    member_results_array = NULL;
  }

  if (member_organism_class_array)
  {
    delete[] member_organism_class_array;
    member_organism_class_array = NULL;
  }

  member_n_survivors = 0;
  member_n_population = 0;
}


void GA_POPULATION_CLASS::function_select(bool s_is_reducible)
{
  int i,k,s_idx,s_half;

  MSORTIDX_SORT_CLASS s_sort_class;
```

```
// -------------------------------
// Calculate the fitness.
// -------------------------------

for(i=0;i<member_n_population;i++)
{
  member_results_array[i] =
  member_organism_class_array[i].
  function_target(s_is_reducible);
}

// -------------------------------
// End of 'Calculate the fitness'.
// -------------------------------


// ---------------------------------------------------------
// Sort the population with minimum target first.
// ---------------------------------------------------------

if (!s_sort_class.
     function_sort(member_results_array,
                   member_n_population))
{
  return; // Error.
}

// ---------------------------------------------------------
// End of 'Sort the population with minimum target first'.
// ---------------------------------------------------------

// -----------------------------------------------------
// Population degeneracy prevention algorithm.
// In large numbers it is equivalent to brut force
// search.
// -----------------------------------------------------

// Sort the first survivors/2 by target and be difference
// from leading. This algorithm promotes some gene diversity.

s_half = member_n_survivors >> 1;

s_idx = s_sort_class.member_array[0].member_index;

member_organism_class_array[member_n_population] =
member_organism_class_array[s_idx];

member_organism_class_array[s_idx].function_print_console();

printf(" %.2lf",member_results_array[s_idx]);

member_results_array[member_n_population] =
member_results_array[s_idx];

for(i=1;i<s_half;i++)
{
```

```
  double s_grade,s_min=0;
  int j,s_min_idx = -1;

  for(j=0;j<member_n_population;j++)
  {
    s_grade = member_results_array[j];

    for(k=0;k<i;k++)
    {

      if (member_organism_class_array[member_n_population + k] ==
          member_organism_class_array[j])
      {
        // Punish organisms for being too similar and thus
        // make the search algorithm more brut force than
        // ordinary evolution.
        // Ordinarry evolution poorly performs and thus the
        // following helps evolution to reach new genes by
        // external control.

        s_grade *= 1.1; // Punish for identity.
      }
    }

    if (s_grade < s_min || s_min_idx < 0)
    {
      s_min = s_grade;
      s_min_idx = j;
    }
  }

  member_organism_class_array[member_n_population+i] =
  member_organism_class_array[s_min_idx];

  member_results_array[member_n_population+i] =
  member_results_array[s_min_idx];
}

// -----------------------------------------------------
// End of 'Population degeneracy prevention algorithm'.
// -----------------------------------------------------

// -------------------------------------------------------------
// The rest survivors half is according to target alone.
// -------------------------------------------------------------

for(i=s_half;i<member_n_survivors;i++)
{
  s_idx = s_sort_class.member_array[i].member_index;

  member_organism_class_array[member_n_population+i] =
  member_organism_class_array[s_idx];

  // member_results_array[member_n_population+i] =
  // member_results_array[s_idx];
}
```

```
// ------------------------------------------------------------
// End of 'The rest survivors half is according to target alone'.
// ------------------------------------------------------------

// ------------------------------------------------------------
// Load the survivors from the upper part of the array.
// This part of the array is used for temporary storage.
// ------------------------------------------------------------

for(i=0;i<member_n_survivors;i++)
{
  member_organism_class_array[i] =
  member_organism_class_array[member_n_population+i];
}

// ------------------------------------------------------------
// End of 'Load the survivors from the upper part of the array'.
// ------------------------------------------------------------

for(i=0;i<member_n_survivors;i++)
{

  // ------------------------
  // Select a mate.
  // ------------------------

  do
  {
    k =
    RAND_RANDOM_CLASS::function_random(member_n_survivors);
  }
  while(i==k);

  // ------------------------
  // End of 'Select a mate'.
  // ------------------------

  // --------------------
  // Reproduce.
  // --------------------

  // -----------------------------------
  // Offspring 1 of indices i,k.
  // -----------------------------------

  s_idx = member_n_survivors + i;

  member_organism_class_array[s_idx] =
  member_organism_class_array[i];

  member_organism_class_array[s_idx].
  function_cross(member_organism_class_array[k]);

  member_organism_class_array[s_idx].function_mutate();

  // -----------------------------------
  // End of 'Offspring 1 of indices i,k'.
```

```
// ------------------------------------

// ------------------------------------
// Offspring 2 of indices i,k.
// ------------------------------------

s_idx += member_n_survivors;

member_organism_class_array[s_idx] =
member_organism_class_array[i];

member_organism_class_array[s_idx].
function_cross(member_organism_class_array[k]);

member_organism_class_array[s_idx].function_mutate();

// ------------------------------------
// End of 'Offspring 2 of indices i,k'.
// ------------------------------------

// ------------------------------------
// Offspring 3 of indices i,k.
// ------------------------------------

s_idx += member_n_survivors;

member_organism_class_array[s_idx] =
member_organism_class_array[i];

member_organism_class_array[s_idx].
function_cross(member_organism_class_array[k]);

member_organism_class_array[s_idx].function_mutate();

// ------------------------------------
// End of 'Offspring 3 of indices i,k'.
// ------------------------------------

#ifdef GA_ALLOW_DEATH_OF_LEADING

// ------------------------------------
// Offspring 4 of indices i,k.
// ------------------------------------

s_idx += member_n_survivors;

member_organism_class_array[s_idx] =
member_organism_class_array[i];

member_organism_class_array[s_idx].
function_cross(member_organism_class_array[k]);

member_organism_class_array[s_idx].function_mutate();

// ------------------------------------
// End of 'Offspring 4 of indices i,k'.
// ------------------------------------
```

```
    #endif

    // -------------------
    // End of 'Reproduce'.
    // -------------------

  }

  #ifdef GA_ALLOW_DEATH_OF_LEADING

  s_idx = member_n_survivors << 2;

  for(i=0;i<member_n_survivors;i++)
  {
    member_organism_class_array[i] =
    member_organism_class_array[i+s_idx];
  }

  #endif
}


void GA_POPULATION_CLASS::function_run(int r_epochs,
                                       bool r_is_reducible,
                                       int r_n_survivors)
{
  int i;

  if (r_epochs>10000000) return;

  if (!function_init(r_n_survivors)) return;

  for(i=0;i<r_epochs;i++)
  {
    printf("%d) ",i+1);
    function_select(r_is_reducible);
    putchar('\r');
    putchar('\n');
    if (_kbhit())
    {
      int r_c;
      r_c = _getch();
      if ('s'==r_c || 'S'==r_c || 'n'==r_c || 'N'==r_c) break;

    }
  }
}

// *******************************************************
// ************ End of 'Population class' ****************
// *******************************************************

// *******************************************************
// ************ Main function ****************************
// *******************************************************
```

```
void main(int ma_ac,char *ma_av[])
{
  int ma_epochs;
  bool ma_reducible;
  int ma_survivors;
  GA_POPULATION_CLASS ma_population_class;

  if (ma_ac!=4 && ma_ac!=3 && ma_ac!=1)
  {
    puts("Useage: GA [Number_Of_Epochs] "
         "[Is_Reducible(0 or 1)] [Survivors]");

    puts("Useage: GA 10000 0 80");

    puts("or");

    puts("Useage: GA [Number_Of_Epochs] [Survivors]");

    puts("Useage: GA 10000 80");

    puts("or");

    puts("Useage: GA");

    puts("Press Enter to exit.");

    getchar();

    return;
  }

  if (4==ma_ac)
  { ma_epochs = atoi(ma_av[1]);
    ma_reducible = atoi(ma_av[2])!=0;
    ma_survivors = atoi(ma_av[3]);
  }
  else   if (3==ma_ac)
  { ma_epochs = atoi(ma_av[1]);
    ma_reducible = false;
    ma_survivors = atoi(ma_av[2]);
  }
  else
  { ma_epochs = 10000;
    ma_reducible = false;
    ma_survivors = 80;
  }

  ma_population_class.function_run(ma_epochs,
                                   ma_reducible,
                                   ma_survivors);

}
```

MSORTIDX.CPP

```
// #include <stdlib.h> // Was used for some QA testing.
#include <stdio.h>
// #include <time.h> // Was used for some QA testing.
// #include <windows.h> // Was used for some QA testing.

#include "msortidx.h"

// ********************************************************
// ********* Merge Sort Class *****************************
// ********************************************************


MSORTIDX_SORT_CLASS::MSORTIDX_SORT_CLASS(void)
{
  member_array=NULL;
  member_auxiliary=NULL;
  member_length=0;

}

MSORTIDX_SORT_CLASS::~MSORTIDX_SORT_CLASS()
{ if (member_array) delete[] member_array;
  if (member_auxiliary) delete[] member_auxiliary;
}

bool MSORTIDX_SORT_CLASS::
     function_init(double *i_arr, // Possibly NULL.
                   int i_len)
{
  int i;

  function_free();

  member_array = new MSORTIDX_ENTRY[i_len];
  member_auxiliary = new MSORTIDX_ENTRY[i_len];

  if (!member_array || !member_auxiliary)
  {
    printf("Failed to allocate %d indices for sorting.\n",i_len);

    if (member_array) delete[] member_array;
    if (member_auxiliary) delete[] member_auxiliary;
    member_array=NULL;
    member_auxiliary=NULL;
    return false;
  }

  member_length = i_len;

  if (i_arr)
  {
    for(i=0;i<i_len;++i)
```

```
    { member_array[i].member_value=i_arr[i];
      member_array[i].member_index=i;
    }
  }

  return true;
}

void MSORTIDX_SORT_CLASS::function_free(void)
{
  if (member_array) delete[] member_array;
  if (member_auxiliary) delete[] member_auxiliary;
  member_array=NULL;
  member_auxiliary=NULL;
  member_length=0;
}


void MSORTIDX_SORT_CLASS::
     function_merge_sorted_arrays(MSORTIDX_ENTRY *msa_arr1,
                                  int msa_len1,
                                  MSORTIDX_ENTRY *msa_arr2,
                                  int msa_len2)
{
  int i,msa_idx1,msa_idx2,msa_index;

  msa_idx1=0;
  msa_idx2=0;
  msa_index=0;

  for(;;msa_index++)
  {
    if (msa_idx1<msa_len1)
    {
      if (msa_idx2<msa_len2)
      {
        if (msa_arr2[msa_idx2].member_value>=
            msa_arr1[msa_idx1].member_value)
        {
          member_auxiliary[msa_index]=msa_arr1[msa_idx1];
          ++msa_idx1;
        }
        else
        { member_auxiliary[msa_index]=msa_arr2[msa_idx2];
          ++msa_idx2;
        }
      }
      else
      {
        member_auxiliary[msa_index]=msa_arr1[msa_idx1];
        ++msa_idx1;
      }
    }
    else
    {
      if (msa_idx2<msa_len2)
      {
```

```
        member_auxiliary[msa_index]=msa_arr2[msa_idx2];
        ++msa_idx2;
       }
       else break;
     }
   }

   for(i=0;i<msa_index;i++)
   {
     msa_arr1[i]=member_auxiliary[i];
   }
}

void MSORTIDX_SORT_CLASS::
     function_merge_sort(int ms_start,int ms_end)
{
  int ms_mid;

  if (ms_start>=ms_end) return;

  ms_mid=(ms_start+ms_end)>>1;

  function_merge_sort(ms_start,ms_mid);
  function_merge_sort(ms_mid+1,ms_end);

  function_merge_sorted_arrays(member_array+ms_start,
                               ms_mid-ms_start+1,
                               member_array+ms_mid+1,
                               ms_end-ms_mid);


}

bool MSORTIDX_SORT_CLASS::
     function_sort(double *s_arr,
                   int s_len)
{
  if (s_len<=0) return false;
  if (!function_init(s_arr,s_len)) return false;

  function_merge_sort(0,s_len-1);

  if (member_auxiliary)
  {
    delete[] member_auxiliary;
    member_auxiliary=NULL;
  }

  return true;
}

// ************************************************************
// ********* End of, "Merge Sort Class" ***********************
// ************************************************************

// QA tests.
```

```
/*
void main(void) // Turned off in a GA program.
{
  MSORTIDX_ENTRY *m_array;
  int i,*m_indices;
  MSORTIDX_SORT_CLASS m_merge_sort;

  int m_count[128];

  m_array = new MSORTIDX_ENTRY[1000];

  if (!m_array )
  {
    return;
  }

  m_indices = new int[1000];

  if (!m_indices)
  {
    delete[] m_array;
    return;
  }

  srand(clock());

  for(i=0;i<128;i++) m_count[i]=0;

  for(i=0;i<1000;i++)
  {
    m_array[i].member_value=rand()>>8;
    m_array[i].member_index=i;
    m_count[(int)m_array[i].member_value]++;

    printf("%ld\n",m_array[i]);
  }

  printf("-----------------------------------\n");

  m_merge_sort.function_sort(m_array,1000);

  for(i=0;i<128;i++)
  {
    printf("count(%d)=%d\n",i,m_count[i]);
  }

  printf("Press Enter to continue.\n");
  printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n");

  getchar();

  for(i=0;i<1000;i++)
  {
    m_count[(int)m_merge_sort.member_array[i].member_value]--;

    printf("(%d) %ld\n",i+1,m_merge_sort.member_array[i]);
```

```
    if (i)
    if (m_merge_sort.member_array[i].member_value<
        m_merge_sort.member_array[i-1].member_value)
    {
      printf("Sorting error\n");
      printf("Press Enter to continue.\n");
      getchar();
      break;
    }
  }

  for(i=0;i<128;i++)
  { if (m_count[i])
    { printf("Sorting error.\n");
      break;
    }
  }

  if (128==i) printf("Sorting is OK\n");
  puts("Press Enter to continue.");
  getchar();

  for(i=0;i<1000;i++) m_indices[i]=0;

  for(i=0;i<1000;i++)
  {
    m_indices[m_merge_sort.member_array[i].member_index]++;

    printf("Index[%d]=%d\n",
           i,m_merge_sort.member_array[i].member_index);
  }

  for(i=0;i<1000;i++)
  {
    if (m_indices[i]!=1)
    {
      puts("Index error.");
      break;
    }
  }

  puts("Press Enter to continue.");
  getchar();

  delete [] m_indices;
  delete [] m_array;
}
*/
```

RAND.CPP

```
#include <windows.h>
#include <stdio.h>

#include "rand.h"
```

```
// 1/(RAND_MAX+1)
#define RAND_SMALL 0.000030517578125

RAND_RANDOM_CLASS::RAND_RANDOM_CLASS(void)
{
  SYSTEMTIME C_time;
  __int64 C_z,C_aux;

  member_bool_selection_array=NULL;
  member_n=0;

  // -------------------------------------
  // Make it depend on the date.
  // -------------------------------------

  GetSystemTime(&C_time);

  C_aux=C_time.wYear;
  C_aux*=12*31*24*60*60;
  C_z=C_aux;
  C_aux=C_time.wMonth;
  C_aux*=31*24*60*60;
  C_z+=C_aux;

  C_aux=C_time.wDay;
  C_aux*=24*60*60;
  C_z+=C_aux;

  C_aux=C_time.wHour;
  C_aux*=60*60;
  C_z+=C_aux;

  C_aux=C_time.wMinute;
  C_aux*=60;
  C_z+=C_aux;

  C_aux=C_time.wSecond;
  C_z+=C_aux;

  // -------------------------------------
  // End of, 'Make it depend on the date'.
  // -------------------------------------

  C_z+=GetTickCount();
  srand((int)C_z);
}

RAND_RANDOM_CLASS::~RAND_RANDOM_CLASS()
{
  function_free();
}

void RAND_RANDOM_CLASS::function_free(void)
{
  if (member_bool_selection_array)
  {
```

```
    delete[] member_bool_selection_array;
    member_bool_selection_array=NULL;
  }

  member_n=0;
}


int RAND_RANDOM_CLASS::function_init(int i_n)
{
  if (i_n==member_n) return 1;

  if (i_n<2) return -4; // Invalid parameter.

  function_free();

  member_bool_selection_array = new bool[i_n];

  if (!member_bool_selection_array) return -1;

  member_n=i_n;

  return 1;
}


int RAND_RANDOM_CLASS::function_random(int r_in)
{
  double r_ret,r_x;
  int r_ret_i;

  if (r_in<2) return 0;

  r_x=rand();
  r_x*=RAND_SMALL;

  r_ret=r_x*r_in;

  r_x=rand();
  r_x*=RAND_SMALL;

  r_ret+=r_x*RAND_SMALL*r_in;

  r_ret_i=(int)r_ret;

  if (r_ret_i>=r_in) r_ret_i=r_in-1;

  return r_ret_i;
}


// Carefully use this function for 2,4,8,16,32,64,128 etc.

int RAND_RANDOM_CLASS::function_rand2(int r_power_of_2)
{
  int r_ret,r_step;
```

```
  r_ret = rand() & (r_power_of_2-1);

  r_step = r_power_of_2 >> RAND_BITS;

  if (r_step)
  {
    r_ret += (rand()&(r_step-1))<<RAND_BITS;
  }

  return r_ret;
}


// Fill member_bool_selection_array with half 1s and half 0s.
int RAND_RANDOM_CLASS::function_select_n_out_of_2n(int snoo2_n)
{
  int i,snoo2_half;
  int *snoo2_arr;

  if (!snoo2_n && !member_n) return -4;

  if (snoo2_n!=member_n)
  {
    int snoo2_ret;
    snoo2_ret=function_init(snoo2_n);
    if (snoo2_ret<0) return snoo2_ret;
  }

  snoo2_arr = new int[snoo2_n];

  if (!snoo2_arr)
  {
    function_free();
    return -1;
  }

  for(i=0;i<snoo2_n;i++)
  { snoo2_arr[i]=i;
    member_bool_selection_array[i] = false;
  }

  snoo2_half = snoo2_n >> 1;

  for(i=0;i<snoo2_half;i++)
  {
    int k;
    int snoo2_swap;

    k = function_random(snoo2_n-i) + i;

    snoo2_swap = snoo2_arr[k];

    snoo2_arr[k] = snoo2_arr[i];

    snoo2_arr[i] = snoo2_swap;

    member_bool_selection_array[snoo2_swap] = true;
```

```
  }

  delete[] snoo2_arr;

  return 1;
}


int RAND_RANDOM_CLASS::function_select_m_out_of_n(int smoo2_m,
                                                  int smoo2_n)
{
  int i;
  int *smoo2_arr;

  if (!smoo2_n && !member_n) return -4;

  if (smoo2_n!=member_n)
  {
    int smoo2_ret;
    smoo2_ret=function_init(smoo2_n);
    if (smoo2_ret<0) return smoo2_ret;
  }

  smoo2_arr = new int[smoo2_n];

  if (!smoo2_arr)
  {
    function_free();
    return -1;
  }

  if (smoo2_m>=smoo2_n)
  {
    delete[] smoo2_arr;

    for(i=0;i<smoo2_n;i++)
    {
      member_bool_selection_array[i] = true;
    }
    return 1;
  }

  for(i=0;i<smoo2_n;i++)
  { smoo2_arr[i]=i;
    member_bool_selection_array[i] = false;
  }

  if (!smoo2_m)
  { delete[] smoo2_arr;
    return 1;
  }

  for(i=0;i<smoo2_m;i++)
  {
    int k;
    int smoo2_swap;
```

```
    k = function_random(smoo2_n-i) + i;

    smoo2_swap = smoo2_arr[k];

    smoo2_arr[k] = smoo2_arr[i];

    smoo2_arr[i] = smoo2_swap;

    member_bool_selection_array[smoo2_swap] = true;
  }

  delete[] smoo2_arr;

  return 1;
}

int RAND_RANDOM_CLASS::
    function_mix(int **m_arr,int m_n)
{
  int i,m_top;

  if (!m_n) return -4;
  if (!m_arr) return -4;

  if (!*m_arr) *m_arr = new int[m_n];

  if (!*m_arr) return -1;

  for(i=0;i<m_n;i++) (*m_arr)[i]=i;

  m_top=m_n-1;

  for(i=0;i<m_top;i++)
  {
    int k;
    int m_swap;

    k = function_random(m_n-i) + i;

    m_swap = (*m_arr)[k];

    (*m_arr)[k] = (*m_arr)[i];

    (*m_arr)[i] = m_swap;

  }

  // User must later delete *m_arr;

  return 1;
}


// inline int RAND_RANDOM_CLASS::operator [](int c_idx)
// {
//   if (!member_bool_selection_array) return -2; // Not found;
//   if (c_idx<0 || c_idx>=member_n) return -4;
```

```
//   return member_bool_selection_array[c_idx];
// }


/*
// Part of the Quality Assurance Tests of the random combinatorics.
void main(void) // Disabled in a GA program.
{
  int i,k;
  int m_count_arr[86];
  int m_r;
  int m_max,m_min,m_min_idx,m_max_idx;
  RAND_RANDOM_CLASS m_rnd;
  int *m_perm;

  m_rnd.function_init(86);

  for(i=0;i<86;i++) m_count_arr[i]=0;

  for(i=0;i<860000;i++)
  {
    m_r=m_rnd.function_random(86);
    m_count_arr[m_r]++;

    // printf("%d ",m_r);
    // if (0==(i%10)) printf("\n");
  }

  printf("\n");

  m_max=m_min=m_count_arr[1];
  m_min_idx=m_max_idx=0;

  for(i=0;i<86;i++)
  {
    printf("Arr[%d]=%d ",
           i,m_count_arr[i]);

    if (0==((i+1)%5)) printf("\n");

    if (m_count_arr[i]<m_min)
    { m_min=m_count_arr[i];
      m_min_idx=i;
    }
    if (m_count_arr[i]>m_max)
    { m_max=m_count_arr[i];
      m_max_idx=i;
    }

  }

  printf("\nMax %d-%d, Min %d-%d \n",
         m_max,m_max_idx,m_min,m_min_idx);

  m_rnd.function_select_n_out_of_2n(86);

  for(i=0;i<86;i++)
```

```
  {
    printf("%1d",
           m_rnd[i]);

    if (0==((i+1)%9)) printf("\n");
  }

  printf("\n");

  for(k=0;k<20;k++)
  {
    m_rnd.function_mix(&m_perm,8);

    for(i=0;i<8;i++)
    {
      printf("%d, ",
             m_perm[i]);

    }

    printf("\n");

    if (m_perm) delete[] m_perm;
  }
}

*/
```

MSORTIDX.H

```
#define NULL 0

struct MSORTIDX_ENTRY
{
  double member_value;
  int member_index;
};


// ************************************************************
// ********* Merge Sort Class *********************************
// ************************************************************

class MSORTIDX_SORT_CLASS
{
  private:

  MSORTIDX_ENTRY *member_auxiliary;

  void function_merge_sorted_arrays(MSORTIDX_ENTRY *msa_arr1,
                                    int msa_len1,
                                    MSORTIDX_ENTRY *msa_arr2,
                                    int msa_len2);
```

```
  void function_merge_sort(int ms_start,int ms_end);

  bool function_init(double *i_arr,
                     int i_len);

  public:

  int member_length;
  MSORTIDX_ENTRY *member_array;

  MSORTIDX_SORT_CLASS::MSORTIDX_SORT_CLASS(void);
  MSORTIDX_SORT_CLASS::~MSORTIDX_SORT_CLASS();


  void function_free(void);
  bool function_sort(double *s_arr,
                     int s_len);

};

// ************************************************************
// ************************************************************
```

RAND.H

```
#define RAND_HALF ((RAND_MAX+1)>>1)
#define RAND_SPAN (RAND_MAX+1)
#define RAND_BITS 15

class RAND_RANDOM_CLASS
{

  // private:

  // // bool *member_bool_selection_array;
  int member_n;

  public:

  bool *member_bool_selection_array;

  RAND_RANDOM_CLASS(void);
  ~RAND_RANDOM_CLASS();
  void function_free(void);
  int function_init(int i_n);
  int function_init1(int i_n);
  static int function_random(int r_x);
  static int function_rand2(int r_power_of_2);
  int function_select_n_out_of_2n(int snoo2_n);
  int function_select_m_out_of_n(int smoo2_m,
                                 int smoo2_n);

  static int function_mix(int **m_arr,int m_n);
  // inline int operator [](int c_idx);
};
```