Projet Modélisation et vérification des systèmes informatiques
TP MALG
par Dominique Méry
6 avril 2017

Le projet MALG reprend les notions enseignées dans le cours MALG. L'idée est d'analyser des programems écrits en C et d'en déduire des propriétés validées à l'aide de la plateforme toolbox TLA et de l'outil Framac. Chaque groupe prendra trois programmes et les analysera selon deux méthodes, en visant la correction partielle et l'absence d'erreurs à l'exécution :

— avec la méthode de traduction des algorithmes en PlusCal et la plateforme Toolbox TLAPS

— avec l'outil Framac

Chaque groupe aura trois algorithmes spécifiques. Il les traduira en un algorithme PlusCal équivalent, vérifiera la correction partielle et l'absence d'erreurs à l'exécution. Il se peut que vous trouviez une optimisation à parrtir de votre analyse et dans ce cas, vous indiquerez comment la méthode vous a permis de le faire.

Le rapport à rendre devra expliquer clairement les opérations réalisées. Ce rapport sera synthétique et mettra en avant les difficultés rencontrées et donnera des possibilités d'extension de cet outil. Le dossier final sera une archive contenant les différents éléments concernant la vérification avec chaque outil et le rapport final en pdf.

La date de remise des dossiers est le 9 mai 2017 et chaque groupe présentera son travail dans les jours suivants. Un dossier est une archive dont le nom est de la forme nom1-nom2-nom3-nom4-projet.zip (utilisez zip pour compresser). Le sujet du message sera tpmalg.

Un groupe est constitué de 4 personnes et vous êtes 52. Il y aura donc 13 groupes.

Le fichier excel groupmalg donne la liste des élèves; les délégués renverront la liste des groupes avec un numéro entre 1 et 13. Cette liste sera renvoyée à Dominique Méry dès que possible.

Table des matières

1 Groupe 1

1.1 Programme premier

```
#include<stdio.h>

int main()
{
   int n, i = 3, count, c;

   printf("Enter␣the␣number␣of␣prime␣numbers␣required\n");
   scanf("%d",&n);

   if ( n >= 1 )
   {
      printf("First␣%d␣prime␣numbers␣are␣:\n",n);
      printf("2\n");
   }

   for ( count = 2 ; count <= n ;   )
   {
      for ( c = 2 ; c <= i - 1 ; c++ )
      {
         if ( i%c == 0 )
            break;
      }
      if ( c == i )
      {
         printf("%d\n",i);
         count++;
      }
      i++;
   }

   return 0;
}
```

1.2 Tri 1 : selection

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

```
int main()
{
   int array[100], n, c, d, position, swap;

   printf("Enter␣number␣of␣elements\n");
   scanf("%d", &n);

   printf("Enter␣%d␣integers\n", n);

   for ( c = 0 ; c < n ; c++ )
      scanf("%d", &array[c]);

   for ( c = 0 ; c < ( n - 1 ) ; c++ )
   {
      position = c;

      for ( d = c + 1 ; d < n ; d++ )
      {
         if ( array[position] > array[d] )
            position = d;
      }
      if ( position != c )
      {
         swap = array[c];
         array[c] = array[position];
         array[position] = swap;
      }
   }

   printf("Sorted␣list␣in␣ascending␣order:\n");

   for ( c = 0 ; c < n ; c++ )
      printf("%d\n", array[c]);

   return 0;
}
```

1.3 Autre algorithme

```
#include <stdio.h>

int p1(int x)
{int a,b,c,i,r,fin;
  a=1;
  b=1;
  i=2;
  if (x==0) { r=1;}
  else { if (x==1) {r=1;}
    else {
      while (i-1 < x)
        {i=i+1;
          c=a;
          a=b;
          b=2*c+2*b;
        } }
    r=b;
  }
  return(r);
}



int main()
{
  int v;
  printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
  scanf("%d", &v);
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d\n",v,p1(v));
  return 0;
}
```

2 Groupe 2

2.1 Programme Amstrong

Armstrong number c program : c programming code to check whether a number is Armstrong or not. Armstrong number is a number which is equal to sum of digits raise to the power total number of digits in the number. Some Armstrong numbers are : 0, 1, 2, 3, 153, 370, 407, 1634, 8208 etc. Read more about Armstrong numbers at Wikipedia. We will consider base 10 numbers in our program. Algorithm to check Armstrong is : First we calculate number of digits in our program and then compute sum of individual digits raise to the power number of digits. If this sum equals input number then number is Armstrong otherwise not an Armstrong number.

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

```
int power(int, int);

int main()
{
   int n, sum = 0, temp, remainder, digits = 0;

   printf("Input␣an␣integer\n");
   scanf("%d", &n);

   temp = n;
   // Count number of digits
   while (temp != 0) {
      digits++;
      temp = temp/10;
   }

   temp = n;

   while (temp != 0) {
      remainder = temp%10;
      sum = sum + power(remainder, digits);
      temp = temp/10;
   }

   if (n == sum)
      printf("%d␣is␣an␣Armstrong␣number.\n", n);
   else
      printf("%d␣is␣not␣an␣Armstrong␣number.\n", n);

   return 0;
}

int power(int n, int r) {
   int c, p = 1;

   for (c = 1; c <= r; c++)
      p = p*n;

   return p;
}
```

2.2 Tri 2 : insertion

```
/* insertion sort ascending order */

#include <stdio.h>

int main()
{
  int n, array[1000], c, d, t;

  printf("Enter␣number␣of␣elements\n");
  scanf("%d", &n);

  printf("Enter␣%d␣integers\n", n);

  for (c = 0; c < n; c++) {
    scanf("%d", &array[c]);
  }

  for (c = 1 ; c <= n - 1; c++) {
    d = c;

    while ( d > 0 && array[d] < array[d-1]) {
      t          = array[d];
      array[d]   = array[d-1];
      array[d-1] = t;

      d--;
    }
  }

  printf("Sorted␣list␣in␣ascending␣order:\n");

  for (c = 0; c <= n - 1; c++) {
    printf("%d\n", array[c]);
  }

  return 0;
}
```

2.3 Autre algorithme

```
#include <stdio.h>

int p2(int x)
{ int z=0,v=0,w=1,t=3,u=0;
```

```
  while (u <  x)
    {
           z=z+v+w;
           v=v+t;
           t=t+6;
           w=w+3;
           u=u+1;
    };
  return (7);
}

int main()
{
  int v,r;
  printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
  scanf("%d", &v); r= v*v*v;
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d␣et␣devrait␣valeur␣%d\n",v,
  return 0;
}
```

3 Groupe 3

3.1 Triangle de Floyd

```
C program to print Floyd's triangle:- This program prints Floyd's triangle. Number
1
2 3
4 5 6
7 8 9 10
It's clear that in Floyd's triangle nth row contains n numbers.
```

```
#include <stdio.h>

int main()
{
  int n, i,  c, a = 1;

  printf("Enter␣the␣number␣of␣rows␣of␣Floyd's␣triangle␣to␣print\n");
  scanf("%d", &n);

  for (i = 1; i <= n; i++)
  {
    for (c = 1; c <= i; c++)
    {
      printf("%d␣",a);
      a++;
    }
    printf("\n");
  }

  return 0;
}
```

3.2 Tri 3 : Buble Sort

```
* Bubble sort code */

#include <stdio.h>

int main()
{
  int array[100], n, c, d, swap;

  printf("Enter␣number␣of␣elements\n");
  scanf("%d", &n);

  printf("Enter␣%d␣integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  for (c = 0 ; c < ( n - 1 ); c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use < */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }

  printf("Sorted␣list␣in␣ascending␣order:\n");
  for ( c = 0 ; c < n ; c++ )
     printf("%d\n", array[c]);

  return 0;
}
```

3.3 Autre algorithme

```
#include <stdio.h>

int p1(int x)
{int a,b,c,i,r,fin;
  a=1;
  b=1;
  i=2;
  if (x==0) { r=1;}
  else { if (x==1) {r=1;}
    else {
      while (i-1 < x)
        {i=i+1;
          c=a;
          a=b;
          b=2*c+2*b;
        } }
    r=b;
  }
  return(r);
}



int main()
{
  int v;
  printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
  scanf("%d", &v);
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d\n",v,p1(v));
  return 0;
}
```

4 Groupe 4

4.1 Fusion de tableaux

```
#include <stdio.h>

void merge(int [], int, int [], int, int []);

int main() {
  int a[100], b[100], m, n, c, sorted[200];

  printf("Input␣number␣of␣elements␣in␣first␣array\n");
  scanf("%d", &m);

  printf("Input␣%d␣integers\n", m);
  for (c = 0; c < m; c++) {
    scanf("%d", &a[c]);
  }

  printf("Input␣number␣of␣elements␣in␣second␣array\n");
  scanf("%d", &n);

  printf("Input␣%d␣integers\n", n);
  for (c = 0; c < n; c++) {
    scanf("%d", &b[c]);
  }

  merge(a, m, b, n, sorted);

  printf("Sorted␣array:\n");

  for (c = 0; c < m + n; c++) {
    printf("%d\n", sorted[c]);
  }

  return 0;
}

void merge(int a[], int m, int b[], int n, int sorted[]) {
  int i, j, k;

  j = k = 0;

  for (i = 0; i < m + n;) {
    if (j < m && k < n) {
      if (a[j] < b[k]) {
        sorted[i] = a[j];
        j++;
      }
      else {
        sorted[i] = b[k];
        k++;
      }
      i++;
    }
    else if (j == m) {
      for (; i < m + n;) {
        sorted[i] = b[k];
        k++;
        i++;
      }
    }
    else {
      for (; i < m + n;) {
        sorted[i] = a[j];
        j++;
        i++;
      }
    }
  }
}
```

4.2 Anagramme

Anagram in c : c program to check whether two strings are anagrams or not, string is assumed to consist of alphabets only. Two words are said to be anagrams of each other if the letters from one word can be rearranged to form the other word. From the above definition it is clear that two strings are anagrams if all characters in both strings occur same number of times. For example "abc" and "cab" are anagram strings, here every character 'a', 'b' and 'c' occur only one time in both strings. Our algorithm tries to find how many times characters appear in the strings and then comparing their corresponding counts.

```
#include <stdio.h>

int check_anagram(char [], char []);

int main()
{
   char a[100], b[100];
   int flag;
   printf("Enter␣first␣string\n");
   gets(a);

   printf("Enter␣second␣string\n");
   gets(b);

   flag = check_anagram(a, b);

   if (flag == 1)
      printf("\"%s\"␣and␣\"%s\"␣are␣anagrams.\n", a, b);
   else
      printf("\"%s\"␣and␣\"%s\"␣are␣not␣anagrams.\n", a, b);

   return 0;
}

int check_anagram(char a[], char b[])
{
   int first[26] = {0}, second[26] = {0}, c = 0;

   while (a[c] != '\0')
   {
      first[a[c]-'a']++;
      c++;
   }

   c = 0;

   while (b[c] != '\0')
   {
      second[b[c]-'a']++;
      c++;
   }

   for (c = 0; c < 26; c++)
   {
      if (first[c] != second[c])
         return 0;
   }

   return 1;
}
```

4.3 Autre algorithme

```
#include <stdio.h>

int p2(int x)
{ int z=0,v=0,w=1,t=3,u=0;

  while (u <  x)
    {
          z=z+v+w;
          v=v+t;
          t=t+6;
          w=w+3;
          u=u+1;
    };
  return (7);
}

int main()
{
  int v,r;
  printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
  scanf("%d", &v); r= v*v*v;
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d␣et␣devrait␣valeur␣%d\n",v,
  return 0;
}
```

5 Groupe 5

5.1 Recherche

```
  bool jw_search ( int *list, int size, int key, int*& rec )
{
  // Basic sequential search
  bool found = false;
  int i;

  for ( i = 0; i < size; i++ ) {
    if ( key == list[i] )
      break;
  }
  if ( i < size ) {
    found = true;
    rec = &list[i];
  }
  return found;
}

  bool jw_search ( int *list , int size , int key, int*& rec )
{
  // Self-organizing (swap with previous) search
  bool found = false;
  int i;

  for ( i = 0; i < size; i++ ) {
    if ( key == list[i] )
      break;
  }
  // Was it found?
  if ( i < size ) {
    // Is it already the first?
    if ( i > 0 ) {
      int save = list[i - 1];
      list[i - 1] = list[i];
      list[i--] = save;
    }
    found = true;
    rec = &list[i];
  }

  return found;
}
```

5.2 Tri par fusion Mergesort

```
/* Helper function for finding the max of two numbers */
int max(int x, int y)
{
    if(x > y)
    {
        return x;
    }
    else
    {
        return y;
    }
}
```

```
/* left is the index of the leftmost element of the subarray; right is one
 * past the index of the rightmost element */
void merge_helper(int *input, int left, int right, int *scratch)
{
    /* base case: one element */
    if(right == left + 1)
    {
        return;
    }
    else
    {
        int i = 0;
        int length = right - left;
        int midpoint_distance = length/2;
        /* l and r are to the positions in the left and right subarrays */
        int l = left, r = left + midpoint_distance;

        /* sort each subarray */
        merge_helper(input, left, left + midpoint_distance, scratch);
        merge_helper(input, left + midpoint_distance, right, scratch);

        /* merge the arrays together using scratch for temporary storage */
        for(i = 0; i < length; i++)
        {
            /* Check to see if any elements remain in the left array; if so,
             * we check if there are any elements left in the right array; if
             * so, we compare them.  Otherwise, we know that the merge must
             * use take the element from the left array */
            if(l < left + midpoint_distance &&
                    (r == right || max(input[l], input[r]) == input[l]))
            {
                scratch[i] = input[l];
                l++;
            }
            else
            {
                scratch[i] = input[r];
                r++;
            }
        }
        /* Copy the sorted subarray back to the input */
        for(i = left; i < right; i++)
        {
            input[i] = scratch[i - left];
        }
    }
}

/* mergesort returns true on success.  Note that in C++, you could also
 * replace malloc with new and if memory allocation fails, an exception will
 * be thrown.  If we don't allocate a scratch array here, what happens?
 *
 * Elements are sorted in reverse order -- greatest to least */

int mergesort(int *input, int size)
{
    int *scratch = (int *)malloc(size * sizeof(int));
    if(scratch != NULL)
    {
        merge_helper(input, 0, size, scratch);
        free(scratch);
        return 1;
    }
    else
    {
        return 0;
    }
}
```

5.3 Autre algorithme

```
#include <stdio.h>

int p1(int x)
{int a,b,c,i,r,fin;
  a=1;
  b=1;
  i=2;
  if (x==0) { r=1;}
  else { if (x==1) {r=1;}
    else {
      while (i-1 < x)
        {i=i+1;
          c=a;
          a=b;
          b=2*c+2*b;
        } }
    r=b;
  }
  return(r);
}


int main()
{
  int v;
  printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
  scanf("%d", &v);
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d\n",v,p1(v));
  return 0;
}
```

6 Groupe 6

6.1 Palindrome d'un nombre

Palindrome number in c : A palindrome number is a number such that if we reverse it, it will not change. For example some palindrome numbers examples are 121, 212, 12321, -454. To check whether a number is palindrome or not first we reverse it and then compare the number obtained with the original, if both are same then number is palindrome otherwise not. C program for palindrome number is given below.

```
#include <stdio.h>

int main()
{
   int n, reverse = 0, temp;

   printf("Enter␣a␣number␣to␣check␣if␣it␣is␣a␣palindrome␣or␣not\n");
   scanf("%d",&n);

   temp = n;

   while( temp != 0 )
   {
      reverse = reverse * 10;
      reverse = reverse + temp%10;
      temp = temp/10;
   }

   if ( n == reverse )
      printf("%d␣is␣a␣palindrome␣number.\n", n);
```

```
    else
        printf("%d␣is␣not␣a␣palindrome␣number.\n", n);

    return 0;
}
```

6.2 Shell Sort

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

#define SIZE 8

void display(int a[],const int size);
void shellsort(int a[], const int size);

int main()
{
    int a[SIZE] = {5,6,3,1,7,8,2,4};

    printf("−−−␣C␣Shell␣Sort␣Demonstration␣−−−␣\n");

    printf("Array␣before␣sorting:\n");
    display(a,SIZE);

    shellsort(a,SIZE);

    printf("Array␣after␣sorting:\n");
    display(a,SIZE);

    return 0;
}

void shellsort(int a[], const int size)
{
  int i, j, inc, tmp;

  inc = 3;
  while (inc > 0)
  {
    for (i=0; i < size; i++)
    {
      j = i;
      tmp = a[i];
      while ((j >= inc) && (a[j-inc] > tmp))
      {
        a[j] = a[j - inc];
        j = j - inc;
      }
      a[j] = tmp;
    }
    if (inc/2 != 0)
      inc = inc/2;
    else if (inc == 1)
      inc = 0;
    else
      inc = 1;
  }
}

void display(int a[],const int size)
{
    int i;
    for(i = 0; i < size; i++)
        printf("%d␣",a[i]);

    printf("\n");
}
```

6.3 Autre algorithme

```
#include <stdio.h>

int p2(int x)
{ int z=0,v=0,w=1,t=3,u=0;

  while (u <  x)
    {
          z=z+v+w;
          v=v+t;
          t=t+6;
          w=w+3;
          u=u+1;
    };
  return (7);
}

int main()
{
  int v,r;
  printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
  scanf("%d", &v); r= v*v*v;
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d␣et␣devrait␣valeur␣%d\n",v,
  return 0;
}
```

7 Groupe 7

7.1 Conversion

```
#include <stdio.h>

int main()
{
  int n, c, k;

  printf("Enter␣an␣integer␣in␣decimal␣number␣system\n");
  scanf("%d", &n);

  printf("%d␣in␣binary␣number␣system␣is:\n", n);

  for (c = 31; c >= 0; c--)
  {
    k = n >> c;

    if (k & 1)
      printf("1");
    else
      printf("0");
  }

  printf("\n");

  return 0;
}
```

7.2 Radix Sort

```
  #include <stdio.h>

void printArray(int * array, int size){
  int i;
  printf("[ ");
  for (i = 0; i < size; i++)
    printf("%d ", array[i]);
  printf("]\n");
}

int findLargestNum(int * array, int size){

  int i;
  int largestNum = -1;

  for(i = 0; i < size; i++){
    if(array[i] > largestNum)
      largestNum = array[i];
  }

  return largestNum;
}

// Radix Sort
void radixSort(int * array, int size){

  printf("\n\nRunning Radix Sort on Unsorted List!\n\n");

  // Base 10 is used
  int i;
  int semiSorted[size];
  int significantDigit = 1;
  int largestNum = findLargestNum(array, size);

  // Loop until we reach the largest significant digit
  while (largestNum / significantDigit > 0){

    printf("\tSorting: %d's place ", significantDigit);
    printArray(array, size);

    int bucket[10] = { 0 };

    // Counts the number of "keys" or digits that will go into each bucket
    for (i = 0; i < size; i++)
      bucket[(array[i] / significantDigit) % 10]++;

    /**
     * Add the count of the previous buckets,
    * Acquires the indexes after the end of each bucket location in the array
                * Works similar to the count sort algorithm
    **/
    for (i = 1; i < 10; i++)
      bucket[i] += bucket[i - 1];

    // Use the bucket to fill a "semiSorted" array
    for (i = size - 1; i >= 0; i--)
      semiSorted[--bucket[(array[i] / significantDigit) % 10]] = array[i];


    for (i = 0; i < size; i++)
      array[i] = semiSorted[i];

    // Move to next significant digit
    significantDigit *= 10;

    printf("\n\tBucket: ");
    printArray(bucket, 10);
  }
}

int main(){

  printf("\n\nRunning Radix Sort Example in C!\n");
  printf("----------------------------------\n");

  int size = 12;
  int list[] = {10, 2, 303, 4021, 293, 1, 0, 429, 480, 92, 2999, 14};

  printf("\nUnsorted List: ");
  printArray(&list[0], size);

  radixSort(&list[0], size);

  printf("\nSorted List:");
  printArray(&list[0], size);
  printf("\n");

  return 0;
}
```

7.3 Autre algorithme

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

```
int p2(int x)
{ int z=0,v=0,w=1,t=3,u=0;

  while (u <  x)
    {
           z=z+v+w;
           v=v+t;
           t=t+6;
           w=w+3;
           u=u+1;
    };
  return (7);
}

int main()
{
  int v,r;
  printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
  scanf("%d", &v); r= v*v*v;
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d␣et␣devrait␣valeur␣%d\n",v,
  return 0;
}
```

8 Groupe 8

8.1 Programme premier

```
#include<stdio.h>

int main()
{
   int n, i = 3, count, c;

   printf("Enter␣the␣number␣of␣prime␣numbers␣required\n");
   scanf("%d",&n);

   if ( n >= 1 )
   {
      printf("First␣%d␣prime␣numbers␣are␣:\n",n);
      printf("2\n");
   }
   for ( count = 2 ; count <= n ;   )
   {
      for ( c = 2 ; c <= i - 1 ; c++ )
      {
         if ( i%c == 0 )
            break;
      }
      if ( c == i )
      {
         printf("%d\n",i);
         count++;
      }
      i++;
   }

   return 0;
}
```

8.2 Tri 1 : selection

```
#include <stdio.h>

int main()
{
   int array[100], n, c, d, position, swap;

   printf("Enter␣number␣of␣elements\n");
   scanf("%d", &n);

   printf("Enter␣%d␣integers\n", n);

   for ( c = 0 ; c < n ; c++ )
      scanf("%d", &array[c]);

   for ( c = 0 ; c < ( n - 1 ) ; c++ )
   {
      position = c;

      for ( d = c + 1 ; d < n ; d++ )
      {
         if ( array[position] > array[d] )
            position = d;
```

```
        }
        if ( position != c )
        {
           swap = array[c];
           array[c] = array[position];
           array[position] = swap;
        }
    }

    printf("Sorted list in ascending order:\n");

    for ( c = 0 ; c < n ; c++ )
       printf("%d\n", array[c]);

    return 0;
}
```

8.3 Autre algorithme

```
#include <stdio.h>

int p1(int x)
{int a,b,c,i,r,fin;
  a=1;
  b=1;
  i=2;
  if (x==0) { r=1;}
  else { if (x==1) {r=1;}
    else {
      while (i-1 < x)
        {i=i+1;
          c=a;
          a=b;
          b=2*c+2*b;
        } }
    r=b;
  }
  return(r);
}



int main()
{
  int v;
```

```
    printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
    scanf("%d", &v);
    printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d\n",v,p1(v));
   return 0;
}
```

9 Groupe 9

9.1 Programme Amstrong

Armstrong number c program : c programming code to check whether a number is Armstrong or not. Armstrong number is a number which is equal to sum of digits raise to the power total number of digits in the number. Some Armstrong numbers are : 0, 1, 2, 3, 153, 370, 407, 1634, 8208 etc. Read more about Armstrong numbers at Wikipedia. We will consider base 10 numbers in our program. Algorithm to check Armstrong is : First we calculate number of digits in our program and then compute sum of individual digits raise to the power number of digits. If this sum equals input number then number is Armstrong otherwise not an Armstrong number.

```
#include <stdio.h>

int power(int, int);

int main()
{
   int n, sum = 0, temp, remainder, digits = 0;

   printf("Input␣an␣integer\n");
   scanf("%d", &n);

   temp = n;
   // Count number of digits
   while (temp != 0) {
      digits++;
      temp = temp/10;
   }

   temp = n;

   while (temp != 0) {
      remainder = temp%10;
      sum = sum + power(remainder, digits);
```

```
        temp = temp/10;
    }

    if (n == sum)
        printf("%d␣is␣an␣Armstrong␣number.\n", n);
    else
        printf("%d␣is␣not␣an␣Armstrong␣number.\n", n);

    return 0;
}

int power(int n, int r) {
    int c, p = 1;

    for (c = 1; c <= r; c++)
        p = p*n;

    return p;
}
```

9.2 Tri 2 : insertion

```
/* insertion sort ascending order */

#include <stdio.h>

int main()
{
   int n, array[1000], c, d, t;

   printf("Enter␣number␣of␣elements\n");
   scanf("%d", &n);

   printf("Enter␣%d␣integers\n", n);

   for (c = 0; c < n; c++) {
     scanf("%d", &array[c]);
   }

   for (c = 1 ; c <= n - 1; c++) {
     d = c;
```

```
    while ( d > 0 && array[d] < array[d-1]) {
      t          = array[d];
      array[d]   = array[d-1];
      array[d-1] = t;

      d--;
    }
  }

  printf("Sorted␣list␣in␣ascending␣order:\n");

  for (c = 0; c <= n - 1; c++) {
    printf("%d\n", array[c]);
  }

  return 0;
}
```

9.3 Autre algorithme

```
#include <stdio.h>

int p2(int x)
{ int z=0,v=0,w=1,t=3,u=0;

  while (u <  x)
    {
          z=z+v+w;
          v=v+t;
          t=t+6;
          w=w+3;
          u=u+1;
    };
  return (7);
}

int main()
{
  int v,r;
  printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
  scanf("%d", &v); r= v*v*v;
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d␣et␣devrait␣valeur␣%d\n",v,
  return 0;
}
```

10 Groupe 10

10.1 Triangle de Floyd

```
C program to print Floyd’s triangle:- This program prints Floyd’s triangle. Number
1
2 3
4 5 6
7 8 9 10
It’s clear that in Floyd’s triangle nth row contains n numbers.
```

```
#include <stdio.h>

int main()
{
  int n, i,  c, a = 1;

  printf("Enter␣the␣number␣of␣rows␣of␣Floyd's␣triangle␣to␣print\n");
  scanf("%d", &n);

  for (i = 1; i <= n; i++)
  {
    for (c = 1; c <= i; c++)
    {
      printf("%d␣",a);
      a++;
    }
    printf("\n");
  }

  return 0;
}
```

10.2 Tri 3 : Buble Sort

```
* Bubble sort code */

#include <stdio.h>

int main()
{
  int array[100], n, c, d, swap;

  printf("Enter␣number␣of␣elements\n");
  scanf("%d", &n);

  printf("Enter␣%d␣integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  for (c = 0 ; c < ( n - 1 ); c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use < */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }

  printf("Sorted␣list␣in␣ascending␣order:\n");

  for ( c = 0 ; c < n ; c++ )
     printf("%d\n", array[c]);

  return 0;
}
```

10.3 Autre algorithme

```
#include <stdio.h>

int p1(int x)
{int a,b,c,i,r,fin;
  a=1;
  b=1;
  i=2;
  if (x==0) { r=1;}
  else { if (x==1) {r=1;}
    else {
      while (i-1 < x)
        {i=i+1;
```

```
            c=a;
            a=b;
            b=2*c+2*b;
          } }
    r=b;
  }
  return(r);
}



int main()
{
  int v;
  printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
  scanf("%d", &v);
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d\n",v,p1(v));
  return 0;
}
```

11 Groupe 11

11.1 Fusion de tableaux

```
#include <stdio.h>

void merge(int [], int, int [], int, int []);

int main() {
  int a[100], b[100], m, n, c, sorted[200];

  printf("Input␣number␣of␣elements␣in␣first␣array\n");
  scanf("%d", &m);

  printf("Input␣%d␣integers\n", m);
  for (c = 0; c < m; c++) {
    scanf("%d", &a[c]);
  }

  printf("Input␣number␣of␣elements␣in␣second␣array\n");
  scanf("%d", &n);

  printf("Input␣%d␣integers\n", n);
  for (c = 0; c < n; c++) {
    scanf("%d", &b[c]);
  }

  merge(a, m, b, n, sorted);

  printf("Sorted␣array:\n");

  for (c = 0; c < m + n; c++) {
    printf("%d\n", sorted[c]);
  }

  return 0;
}

void merge(int a[], int m, int b[], int n, int sorted[]) {
  int i, j, k;

  j = k = 0;

  for (i = 0; i < m + n;) {
    if (j < m && k < n) {
      if (a[j] < b[k]) {
        sorted[i] = a[j];
        j++;
      }
      else {
        sorted[i] = b[k];
        k++;
      }
      i++;
    }
    else if (j == m) {
      for (; i < m + n;) {
        sorted[i] = b[k];
        k++;
        i++;
      }
    }
    else {
      for (; i < m + n;) {
        sorted[i] = a[j];
        j++;
        i++;
      }
      }
    }
}
```

11.2 Anagramme

Anagram in c : c program to check whether two strings are anagrams or not, string is assumed to consist of alphabets only. Two words are said to be anagrams of each other if the letters from one word can be rearranged to form the other word. From the above definition it is clear that two strings are anagrams if all characters in both strings occur same number of times. For example "abc" and "cab" are anagram strings, here every character 'a', 'b' and 'c' occur only one time in both strings. Our algorithm tries to find how many times characters appear in the strings and then comparing their corresponding counts.

```
#include <stdio.h>

int check_anagram(char [], char []);

int main()
{
   char a[100], b[100];
   int flag;

   printf("Enter␣first␣string\n");
   gets(a);

   printf("Enter␣second␣string\n");
   gets(b);

   flag = check_anagram(a, b);

   if (flag == 1)
      printf("\"%s\"␣and␣\"%s\"␣are␣anagrams.\n", a, b);
   else
      printf("\"%s\"␣and␣\"%s\"␣are␣not␣anagrams.\n", a, b);

   return 0;
}

int check_anagram(char a[], char b[])
{
   int first[26] = {0}, second[26] = {0}, c = 0;
   while (a[c] != '\0')
   {
      first[a[c]-'a']++;
      c++;
   }

   c = 0;

   while (b[c] != '\0')
   {
      second[b[c]-'a']++;
      c++;
   }

   for (c = 0; c < 26; c++)
   {
      if (first[c] != second[c])
         return 0;
   }

   return 1;
}
```

11.3 Autre algorithme

```
#include <stdio.h>

int p2(int x)
{ int z=0,v=0,w=1,t=3,u=0;

  while (u <  x)
    {
          z=z+v+w;
          v=v+t;
          t=t+6;
          w=w+3;
          u=u+1;
    };
  return (7);
}

int main()
{
  int v,r;
  printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
  scanf("%d", &v); r= v*v*v;
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d␣et␣devrait␣valeur␣%d\n",v,
  return 0;
}
```

12 Groupe 12

12.1 Recherche

```
  bool jw_search ( int *list, int size, int key, int*& rec )
{
  // Basic sequential search
  bool found = false;
  int i;

  for ( i = 0; i < size; i++ ) {
    if ( key == list[i] )
      break;
  }
  if ( i < size ) {
    found = true;
    rec = &list[i];
  }

  return found;
}

  bool jw_search ( int *list, int size, int key, int*& rec )
{
  // Self-organizing (swap with previous) search
  bool found = false;
  int i;

  for ( i = 0; i < size; i++ ) {
    if ( key == list[i] )
      break;
  }
  // Was it found?
  if ( i < size ) {
    // Is it already the first?
    if ( i > 0 ) {
      int save = list[i - 1];
      list[i - 1] = list[i];
      list[i--] = save;
    }
    found = true;
    rec = &list[i];
  }

  return found;
}
```

12.2 Tri par fusion Mergesort

```
/* Helper function for finding the max of two numbers */
int max(int x, int y)
{
    if(x > y)
    {
        return x;
    }
    else
    {
        return y;
    }
}

/* left is the index of the leftmost element of the subarray; right is one
 * past the index of the rightmost element */
void merge_helper(int *input, int left, int right, int *scratch)
{
    /* base case: one element */
    if(right == left + 1)
    {
        return;
    }
    else
    {
        int i = 0;
        int length = right - left;
        int midpoint_distance = length/2;
        /* l and r are to the positions in the left and right subarrays */
        int l = left, r = left + midpoint_distance;

        /* sort each subarray */
        merge_helper(input, left, left + midpoint_distance, scratch);
        merge_helper(input, left + midpoint_distance, right, scratch);
        /* merge the arrays together using scratch for temporary storage */
        for(i = 0; i < length; i++)
        {
            /* Check to see if any elements remain in the left array; if so,
             * we check if there are any elements left in the right array; if
             * so, we compare them.  Otherwise, we know that the merge must
             * use take the element from the left array */
            if(l < left + midpoint_distance &&
                    (r == right || max(input[l], input[r]) == input[l]))
            {
                scratch[i] = input[l];
                l++;
            }
            else
            {
                scratch[i] = input[r];
                r++;
            }
        }
        /* Copy the sorted subarray back to the input */
        for(i = left; i < right; i++)
        {
            input[i] = scratch[i - left];
        }
    }
}

/* mergesort returns true on success.  Note that in C++, you could also
 * replace malloc with new and if memory allocation fails, an exception will
 * be thrown.  If we don't allocate a scratch array here, what happens?
 *
 * Elements are sorted in reverse order -- greatest to least */

int mergesort(int *input, int size)
{
    int *scratch = (int *)malloc(size * sizeof(int));
    if(scratch != NULL)
    {
        merge_helper(input, 0, size, scratch);
        free(scratch);
        return 1;
    }
    else
    {
        return 0;
    }
}
```

12.3 Autre algorithme

```
#include <stdio.h>

int p1(int x)
{int a,b,c,i,r,fin;
  a=1;
  b=1;
  i=2;
  if (x==0) { r=1;}
  else { if (x==1) {r=1;}
    else {
      while (i-1 < x)
        {i=i+1;
          c=a;
          a=b;
          b=2*c+2*b;
        } }
    r=b;
  }
  return(r);
}



int main()
{
  int v;
  printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
  scanf("%d", &v);
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d\n",v,p1(v));
  return 0;
}
```

13 Groupe 13

13.1 Palindrome d'un nombre

Palindrome number in c : A palindrome number is a number such that if we reverse it, it will not change. For example some palindrome numbers examples are 121, 212, 12321, -454. To check whether a number is palindrome or not first we reverse it and then compare the number obtained with the original, if both *are same then number is palindrome otherwise not. C program for palindrome number is given below.*

```
#include <stdio.h>

int main()
{
   int n, reverse = 0, temp;

   printf("Enter␣a␣number␣to␣check␣if␣it␣is␣a␣palindrome␣or␣not\n");
   scanf("%d",&n);

   temp = n;

   while( temp != 0 )
   {
      reverse = reverse * 10;
      reverse = reverse + temp%10;
      temp = temp/10;
   }

   if ( n == reverse )
      printf("%d␣is␣a␣palindrome␣number.\n", n);
   else
      printf("%d␣is␣not␣a␣palindrome␣number.\n", n);

   return 0;
}
```

13.2 Shell Sort

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

#define SIZE 8

void display(int a[],const int size);
void shellsort(int a[], const int size);

int main()
{
    int a[SIZE] = {5,6,3,1,7,8,2,4};
    printf("---␣C␣Shell␣Sort␣Demonstration␣---␣\n");

    printf("Array␣before␣sorting:\n");
    display(a,SIZE);

    shellsort(a,SIZE);

    printf("Array␣after␣sorting:\n");
    display(a,SIZE);

    return 0;
}

void shellsort(int a[], const int size)
{
  int i, j, inc, tmp;

  inc = 3;
  while (inc > 0)
  {
    for (i=0; i < size; i++)
    {
      j = i;
      tmp = a[i];
      while ((j >= inc) && (a[j-inc] > tmp))
      {
        a[j] = a[j - inc];
        j = j - inc;
      }
      a[j] = tmp;
    }
    if (inc/2 != 0)
      inc = inc/2;
    else if (inc == 1)
      inc = 0;
    else
      inc = 1;
  }
}

void display(int a[],const int size)
{
    int i;
    for(i = 0; i < size; i++)
        printf("%d␣",a[i]);
    printf("\n");
}
```

13.3 Autre algorithme

```
#include <stdio.h>

int p2(int x)
{ int z=0,v=0,w=1,t=3,u=0;

  while (u <  x)
    {
          z=z+v+w;
          v=v+t;
          t=t+6;
          w=w+3;
          u=u+1;
    };
  return (7);
}

int main()
{
  int v,r;
  printf("Entrez␣la␣valeur␣␣pour␣␣v\n");
  scanf("%d", &v); r= v*v*v;
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣p2(%d)=%d␣et␣devrait␣valeur␣%d\n",v,
  return 0;
}
```