Projet Modélisation et vérification des systèmes informatiques
Projet VERIFICATION
par Dominique Méry
1er septembre 2026

Le projet VERIFICATION reprend les notions enseignées dans le cours MALG. L'idée est d'analyser des programmes écrits en C et d'en déduire des propriétés validées à l'aide de l'outil Frama-C. Chaque groupe prendra un des deux programmes affectés au numéro affecté et l'analysera selon la méthode Frama-C, en visant la correction partielle et l'absence d'erreurs à l'exécution.
Chaque groupe aura un algorithme spécifique. 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.
Comme chaque groupe a deux algorithmes ou programmes, celui qui n'aura pas été traité sera spécifié selon ses pré et post conditions.
Le rapport à rendre devra expliquer clairement les opérations réalisées. Ce rapport sera synthétique (moins de quatre pages) 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 et le rapport final en pdf.
La date de remise des dossiers est le **15 novembre 2026** et chaque groupe présentera son travail dans les jours suivants. Un dossier est une archive dont le nom est de la forme nom1-nom2-projet.zip (utilisez zip pour compresser) avec comme sujet "mvsi".
Les groupes seront constitués dès que possible et au plus tard le 15 septembre 2026.
Les élèves enverront un fichier excel avec les différents groupes à Dominique Méry. Pour faciliter les mails et leur traitement, il est plus simple de mettre le sujet « mvsi » lors de l'envoi des courriers relatifs à ce projet.
Les sujets possibles sont donnés dans la suite et chaque groupe choisira un sujet qui devra être différent des autres groupes.

Table des matières

1 Groupe 1 **6**
- 1.1 Triangle de Floyd . . . 6
- 1.2 Bubble Sort . . . 6

2 Groupe 2 **7**
- 2.1 Programme premier . . . 7
- 2.2 Selection Sort . . . 8

3 Groupe 3 **8**
- 3.1 Power5 . . . 8
- 3.2 Fusion de tableaux . . . 12

4 Groupe 4 **13**
- 4.1 Recherche . . . 13
- 4.2 Mergesort . . . 14

5 Groupe 5 **15**
- 5.1 Power4 . . . 15
- 5.2 Bin2dec . . . 16

6 Groupe 6 **17**
- 6.1 powera.c . . . 17
- 6.2 Radix Sort . . . 19

7 Groupe 7 **20**
- 7.1 Binary Search (iterative) . . . 20
- 7.2 GCD SCM . . . 21

8 Groupe 8 **23**
- 8.1 Binary Search (iterative) . . . 23
- 8.2 powerb.c . . . 24

9 Groupe 9 **26**
- 9.1 Sorting . . . 26
- 9.2 Power6bis.c . . . 29

10 Groupe 10 **30**
- 10.1 Quick Sort . . . 30
- 10.2 Power6.c . . . 31

11 Groupe 11 **33**
- 11.1 Shell Sort . . . 33
- 11.2 GCD SCM . . . 34

12 Groupe 12 **35**
- 12.1 Rotation d'un tableau . . . 35
- 12.2 Calcul du min et du max . . . 36

13 Groupe 13 **37**
- 13.1 Calcul des Ã©lÃ©ments communs . . . 37
- 13.2 Fusipn de deux tableaux . . . 38

14 Groupe 14 **39**
- 14.1 Calcul des Ã©lÃ©ments communs . . . 39
- 14.2 Fusipn de deux tableaux . . . 40

15 Groupe 15 **41**
15.1 Suppression d'un Ã©lÃ©ment 41
15.2 VÃ©rification de l'Ã©galitÃ© de deux matrices 41

16 Groupe 16 **42**
16.1 Destruction des duplicas 42
16.2 43

Listings

codes/floyd.c . . . 6
codes/bubblesort.c . . . 6
codes/premier.c . . . 7
codes/selectionsort.c . . . 8
codes/power5.c . . . 8
codes/fusion.c . . . 12
codes/recherche.c . . . 13
codes/mergesort.c . . . 14
codes/power4.c . . . 15
codes/bin2dec.c . . . 17
codes/power2.c . . . 17
codes/power4.c . . . 17
codes/radixsort.c . . . 19
codes/binarysearchiterative.c . . . 20
codes/gcdscm.c . . . 21
codes/gcdscmrun.c . . . 22
codes/gcdscmrunrun.c . . . 22
codes/binarysearchiterative.c . . . 23
codes/power3.c . . . 24
codes/power4.c . . . 25
codes/sorting.c . . . 26
codes/power6bis.c . . . 29
codes/quick0.c . . . 30
codes/power6.c . . . 31
codes/shellsort.c . . . 33
codes/gcdscm.c . . . 34
codes/gcdscmrun.c . . . 34
codes/gcdscmrunrun.c . . . 35
codes/rotation.c . . . 35
codes/minmax.c . . . 36
codes/commonbis.c . . . 37
codes/primes.c . . . 38
codes/common.c . . . 39
codes/merge.c . . . 40
codes/remove.c . . . 41
codes/checkmatrix.c . . . 41
codes/simplify.c . . . 42
codes/average.c . . . 43

Liste des groupes avec les numéros

— Groupe 1
— Groupe 2
— Groupe 3
— Groupe 4
— Groupe 5
— Groupe 6
— Groupe 7

1 Groupe 1

1.1 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;
}
```

1.2 Bubble 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;
}
```

2 Groupe 2

2.1 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;
```

```
}
```

2.2 Selection Sort

```
#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;
}
```

3 Groupe 3

3.1 Power5

```
#include <stdio.h>

int tp(int x)
{int a=0,b=0,c=0,d=0,e=1,f=0,g=0,h=12,i=0, j=4,k=0,l=6,m=4,p=0,  aa,ab,ac,ad,af,ae,

  aa=a;ab=b;ac=c;ad=d;af=f;ae=e;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;
  while (p< x)
```

```
        {
          printf("a=%d  ; b=%d  ; c=%d  ; d = %d ; u = %d\n",a,b,c,d,e);
          aa=a;ab=b;ac=c;ad=d;af=f;ae=e;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;
          a=aa+ab+ac+ad+ae;
          b=ab+af+ai+aj;
          c=ac+ak+al;
          d=ad+am;
          e=ae;
          f=af+ag+ah;
          g = ag + 24;
          h=ah;
          i=ai+12;
          j=aj;
          k = ak +12;
          l = al;
          m = am;
          p=p+1;
          aa=a;ab=b;ac=c;ad=d;af=f;ae=e;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;
        }
          printf("a=%d  ; b=%d  ; c=%d  ; d = %d ; u = %d\n",a,b,c,d,e);
  return(a);
}


int fpower5(int x)

{int a=0,b=0,c=0,d=0,e=1,f=0,g=0,h=5,i=0, j=10,k=10,l=0,m=20,o=30,p=60,z=0,aa,ab,ac

  aa=a;ab=b;ac=c;ad=d;ae=e;af=f;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;ao=o;ap=p;

  while (z< x)
        {
  aa=a;ab=b;ac=c;ad=d;ae=e;af=f;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;ao=o;ap=p;
          a=a+b+c+d+e;
          b=b+f+g+h;
          c=c+i +j;
          d=d+k;
          e = e + 5;
          f=f+l+m;
          g=g+o;
          i=i+o;
          h=h+20;
          j=j+30;
          k=k+20;
          l=l+p;
          m=m+60;
          o=o+60;
          p=p+120;
          z=z+1;
          //      printf("|  %d | %d | %d | %d | %d | %d | %d | %d | %d | %d |
%d\n",a,b,c,d,e,f,g,i,k,h,j,l,m,o,p,z);
  aa=a;ab=b;ac=c;ad=d;ae=e;af=f;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;ao=o;ap=p;
        }
  return(a);
}
```

```
int op5(int x)

{int a=0,b=0,c=0,d=0,e=1,f=0,g=0,h=5, //i=0,
    j=10,k=10,l=0,m=20,o=30,p=60,z=0,aa,ab,ac,ad,ae,af,ag,ai,ah,aj,ak,al,am,ao,ap;

  while (z< x)
        {
          a=a+b+c+d+e;
          b=b+f+g+h;
          //       c=c+i +j;
          c=c+g+j;
          d=d+k;
          e = e + 5;
          f=f+l+m;
          g=g+o;
          //       i=i+o;
          h=h+20;
          j=j+30;
          k=k+20;
          l=l+p;
          m=m+60;
          o=o+60;
          p=p+120;
          z=z+1;
          //       printf("|  %d | %d | %d | %d | %d | %d | %d | %d | %d | %d |
%d\n",a,b,c,d,e,f,g,i,k,h,j,l,m,o,p,z);
        }
  return(a);
}



int p5(int x)

{int a=0,b=0,c=0,d=0,e=1,f=0,g=0,h=5,
    j=10,k=10,l=0,m=20,o=30,p=60,z=0,
    aa,ab,ac,ad,ae,af,ag,ah,aj,ak,al,
    am,ao,ap;

  while (z< x)
        {
          a=a+b+c+d+e;
          b=b+f+g+h;
          c=c+g+j;
          d=d+k;
          e = e + 5;
          f=f+l+m;
          g=g+o;
          h=h+20;
          j=j+30;
          k=k+20;
          l=l+p;
          m=m+60;
          o=o+60;
```

```
            p=p+120;
            z=z+1;
          }
  return(a);
}


int g(int p)

{int  az,z,aa,a,at,t,ay,y,ax,x,aw,w,av,v,au,u,i=0;
  z=0;a=12;t=6;y=4;x=0;w=1;v=0;u=0;z=0;  i=0;
  while (i< p)
          {
            z=z+u+v+w;
            v=v+t;
            t=t+12;
            w=w+4;
            u=u+x+y;
            x=x+a;
            a=a+24;
            y=y+12;
            i=i+1;
          }
  z=z*p;
  return(z);
}



int h(int p)
{int  az,z,aa,a,at,t,ay,y,ax,x,aw,w,av,v,au,u,i=0,ai;
  z=0;a=12;t=6;y=4;x=0;w=1;v=0;u=0;z=0;  i=0;ai=0;
  while (z<= p)
          {
            az= z;ai=i;
            z=z+u+v+w;
            v=v+t;
            t=t+12;
            w=w+4;
            u=u+x+y;
            x=x+a;
            a=a+24;
            y=y+12;
            i=i+1;
          }
  if (i == 0 || i==1) {return(i);}
  else {  return(ai); }

}



int main()
{
  int v;
```

```
  printf("Entrez la valeur 8 pour  v\n");
  scanf("%d", &v);
  // printf(" which is %d --> for power 4 %d et power 5 %d\n",v,g(v),fpower
  printf("%d --> p5=%d  et fpower5  %d\n",v,p5(v),fpower5(v));
  return 0;
}
```

3.2 Fusion de tableaux

```
\begin{lstlisting}


#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++;
      }
    }
  }
}
\end{lstlisting}
```

4 Groupe 4

4.1 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;
}
```

4.2 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 Groupe 5

5.1 Power4

```
#include <stdio.h>

int tp(int p)
{int  az,z,aa,a,at,t,ay,y,ax,x,aw,w,av,v,au,u,i=0;
  z=0;a=12;t=6;y=4;x=0;w=1;v=0;u=0;z=0;
  az=z;aa=a;at=t;ay=y;ax=x;aw=w;av=v;au=u;az=z;i=0;
```

```
  while (i< p)
        {
          printf("z=%d  ; v=%d  ; t=%d  ; w = %d; a = %d; y = %d; x = %d; u = %d;\n
          z=az+au+av+aw;
          v=av+at;
          t=at+12;
          w=aw+4;
          a=aa+24;
          y=ay+12;
          x=ax+aa;
          u=au+ax+ay;
          az=z;aa=a;at=t;ay=y;ax=x;aw=w;av=v;au=u;az=z;
          i=i+1;

        }
  return(z);
}



int tt(int p)
{int  az,z,aa,a,at,t,ay,y,ax,x,aw,w,av,v,au,u,i=0;
  z=0;a=12;t=6;y=4;x=0;w=1;v=0;u=0;z=0;

  i=0;
  while (i< p)
        {
          printf("z=%d  ; v=%d  ; t=%d  ; w = %d; a = %d; y = %d; x = %d; u = %d;\n
          z=z+u+v+w;
          v=v+t;
          t=t+12;
          w=w+4;
          u=u+x+y;
          x=x+a;
          a=a+24;
          y=y+12;
          i=i+1;

        }
  return(z);
}


int main()
{
  int v;

  printf("Entrez la valeur 8 pour  v\n");
  scanf("%d", &v);
  printf(" voici la rÃ©ponse de votre solution %d --> %d\n",v,tt(v));
  return 0;
}
```

5.2 Bin2dec

```
#include <math.h>
#include <stdio.h>
int convert(long long n);
int main() {
    long long n;
    printf("Enter a binary number: ");
    scanf("%lld", &n);
    printf("%lld in binary = %d in decimal", n, convert(n));
    return 0;
}

int convert(long long n) {
    int dec = 0, i = 0, rem;
    while (n != 0) {
        rem = n % 10;
        n /= 10;
        dec += rem * pow(2, i);
        ++i;
    }
    return dec;
}
```

6 Groupe 6

6.1 powera.c

Vous disposez de deux fonctions calculant x^2 et x^4. Ecrire une fonction calculant xv avec ces deux fonctions et faire les vérifications demandées. Les deux fonctions doivent être prouvées aussi mais c 'est surtout les appels qui sont à vérifier. Il faudra utiliser l'équation $x^6 = x^{3^2}$ pour la fonctipn powera.c et aussi $x^6 = x^{2^3}$.

```
#include <stdio.h>


int power2(int x)
{int  r,k,cv,cw,or,ok,ocv,ocw;
  r=0;k=0;cv=0;cw=0;or=0;ok=k;ocv=cv;ocw=cw;
  while (k<x)
        {
          ok=k;ocv=cv;ocw=cw;
          k=ok+1;
          cv=ocv+ocw+1;
          cw=ocw+2;}
  r=cv;return(r);}

int main()
{
  int v;

  printf("Entrez la valeur  pour  v\n");
  scanf("%d", &v);
  printf(" voici la rÃ©ponse de votre solution %d --> %d\n",v,power2(v));
  return 0;
}
```

```
#include <stdio.h>

int tp(int p)
{int  az,z,aa,a,at,t,ay,y,ax,x,aw,w,av,v,au,u,i=0;
  z=0;a=12;t=6;y=4;x=0;w=1;v=0;u=0;z=0;
  az=z;aa=a;at=t;ay=y;ax=x;aw=w;av=v;au=u;az=z;i=0;
  while (i< p)
        {
          printf("z=%d  ; v=%d  ; t=%d  ; w = %d; a = %d; y = %d; x = %d; u = %d;\n
          z=az+au+av+aw;
          v=av+at;
          t=at+12;
          w=aw+4;
          a=aa+24;
          y=ay+12;
          x=ax+aa;
          u=au+ax+ay;
          az=z;aa=a;at=t;ay=y;ax=x;aw=w;av=v;au=u;az=z;
          i=i+1;

        }
  return(z);
}



int tt(int p)
{int  az,z,aa,a,at,t,ay,y,ax,x,aw,w,av,v,au,u,i=0;
  z=0;a=12;t=6;y=4;x=0;w=1;v=0;u=0;z=0;

  i=0;
  while (i< p)
        {
          printf("z=%d  ; v=%d  ; t=%d  ; w = %d; a = %d; y = %d; x = %d; u = %d;\n
          z=z+u+v+w;
          v=v+t;
          t=t+12;
          w=w+4;
          u=u+x+y;
          x=x+a;
          a=a+24;
          y=y+12;
          i=i+1;

        }
  return(z);
}


int main()
{
  int v;

  printf("Entrez la valeur 8 pour  v\n");
  scanf("%d", &v);
```

```
  printf("␣voici␣la␣rÃ©ponse␣de␣votre␣solution␣%d␣-->␣%d\n",v,tt(v));
  return 0;
}
```

6.2 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 Groupe 7

7.1 Binary Search (iterative)

```
#include <stdio.h>

// Iterative implementation of the binary search algorithm to return
// the position of `target` in array `nums` of size `n`

int binarySearch(int nums[], int n, int target)
{
        // search space is nums[low..high]
        int low = 0, high = n - 1;
```

```
        // loop till the search space is exhausted
        while (low <= high)
        {
                // find the mid-value in the search space and
                // compares it with the target

                int mid = (low + high)/2;       // overflow can happen
                // int mid = low + (high - low)/2;
                // int mid = high - (high - low)/2;

                // target value is found
                if (target == nums[mid]) {
                        return mid;
                }

                // if the target is less than the middle element, discard all eleme
                // in the right search space, including the middle element
                else if (target < nums[mid]) {
                        high = mid - 1;
                }

                // if the target is more than the middle element, discard all eleme
                // in the left search space, including the middle element
                else {
                        low = mid + 1;
                }
        }

        // target doesn't exist in the array
        return -1;
}

int main(void)
{
        int nums[] = { 2, 5, 6, 8, 9, 10 };
        int target = 5;

        int n = sizeof(nums)/sizeof(nums[0]);
        int index = binarySearch(nums, n, target);

        if (index != -1) {
                printf("Element␣found␣at␣index␣%d", index);
        }
        else {
                printf("Element␣not␣found␣in␣the␣array");
        }

        return 0;
}
```

7.2 GCD SCM

```
#include <limits.h>
#include  "structure.h"
#include  "gcdscm.h"
```

```
struct paire codegcdscm(int x1,int x2) {
  int y1 = x1;
  int y2 = x2;
  int y3 = x2;
  int y4 = 0;
  struct paire r;
  while (y1 != y2) {
    while (y1 > y2) { y1 = y1 -y2; y4=y4+y3;};
    while (y1 < y2) { y2 = y2 -y1;y3=y3+y4;};
  };
  r.gcd = y1;
  r.scm = y3+y4;
  r.wit= x1*x2;
   return r;
}

#include <stdio.h>
#include "structure.h"
#include "gcdscm.h"

int main () {

  int counter;
    for( counter=0; counter<5; counter++ ) {
      int x1,x2;
      struct paire r;
      printf("Enter a natural number x1:");
      scanf("%d", &x1);
      printf("Enter a natural number x2:");
      scanf("%d", &x2);
      r = codegcdscm(x1,x2);
      printf ("Result: %d,%d ----> %d,%d\n", x1,x2,r.gcd,r.scm);

    };
}

#include <stdio.h>




struct paire {
    unsigned gcd;
    unsigned scm;
  unsigned wit; };

struct paire codegcdscm(int x1,int x2) {
  int y1 = x1;
  int y2 = x2;
  int y3 = x2;
  int y4 = 0;
  struct paire r;
  while (y1 != y2) {
    while (y1 > y2) { y1 = y1 -y2; y4=y4+y3;};
    while (y1 < y2) { y2 = y2 -y1;y3=y3+y4;};
```

```
  };
  r.gcd = y1;
  r.scm = y3+y4;
  r.wit= x1*x2;
   return r;
}

int main () {

  int counter;
    for( counter=0; counter<5; counter++ ) {
      int x1,x2;
      struct paire r;
      printf("Enter a natural number x1:");
      scanf("%d", &x1);
      printf("Enter a natural number x2:");
      scanf("%d", &x2);
      r = codegcdscm(x1,x2);
      printf ("Result: %d,%d ----> %d,%d\n", x1,x2,r.gcd,r.scm);

    };
}
```

8 Groupe 8

8.1 Binary Search (iterative)

```
#include <stdio.h>

// Iterative implementation of the binary search algorithm to return
// the position of 'target' in array 'nums' of size 'n'

int binarySearch(int nums[], int n, int target)
{
        // search space is nums[low..high]
        int low = 0, high = n - 1;

        // loop till the search space is exhausted
        while (low <= high)
        {
                // find the mid-value in the search space and
                // compares it with the target

                int mid = (low + high)/2;       // overflow can happen
                // int mid = low + (high - low)/2;
                // int mid = high - (high - low)/2;

                // target value is found
                if (target == nums[mid]) {
                        return mid;
                }

                // if the target is less than the middle element, discard all eleme
                // in the right search space, including the middle element
                else if (target < nums[mid]) {
                        high = mid - 1;
```

```
                }

                // if the target is more than the middle element, discard all eleme
                // in the left search space, including the middle element
                else {
                        low = mid + 1;
                }
        }

        // target doesn't exist in the array
        return -1;
}

int main(void)
{
        int nums[] = { 2, 5, 6, 8, 9, 10 };
        int target = 5;

        int n = sizeof(nums)/sizeof(nums[0]);
        int index = binarySearch(nums, n, target);

        if (index != -1) {
                printf("Element␣found␣at␣index␣%d", index);
        }
        else {
                printf("Element␣not␣found␣in␣the␣array");
        }

        return 0;
}
```

8.2 powerb.c

Vous disposez de deux fonctions calculant x^3 et x^4. Ecrire une fonction calculant x^7 avec ces deux fonctions et faire les vérifications demandées. Dans ce cas, vous utiliserez la fonction x^3 qui a été prouvée mais qu'il faudra rappeler. Il faudra utiliser l'équation $xx^7 = x^3 \cdot x^4$ pour la fonctip powerb.c

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

int tp(int x)
{int  az,z,v,av,w,aw,t,at,u=0,au;
  z=0;v=0;w=1;t=3; aw=w;az=z;at=t;av=v;au=u;
  while (u< x)
        {
          printf("z=%d␣␣;␣v=%d␣␣;␣w=%d␣␣;␣t␣=␣%d;␣u␣=␣%d\n",z,v,w,t,u);
          au=u;
          z=az+av+aw;
          v=av+at;
          t=at+6;
          w=aw+3;
          aw=w;az=z;at=t;av=v;
          u=u+1;

        }
  return(z);
}
```

```
int main()
{
  int v;

  printf("Entrez la valeur 8 pour  v\n");
  scanf("%d", &v);
  printf(" voici la rÃ©ponse de votre solution %d --> %d\n",v,tp(v));
  return 0;
}

#include <stdio.h>

int tp(int p)
{int  az,z,aa,a,at,t,ay,y,ax,x,aw,w,av,v,au,u,i=0;
  z=0;a=12;t=6;y=4;x=0;w=1;v=0;u=0;z=0;
  az=z;aa=a;at=t;ay=y;ax=x;aw=w;av=v;au=u;az=z;i=0;
  while (i< p)
        {
          printf("z=%d  ; v=%d  ; t=%d  ; w= %d; a = %d; y = %d; x = %d; u = %d;\n
          z=az+au+av+aw;
          v=av+at;
          t=at+12;
          w=aw+4;
          a=aa+24;
          y=ay+12;
          x=ax+aa;
          u=au+ax+ay;
          az=z;aa=a;at=t;ay=y;ax=x;aw=w;av=v;au=u;az=z;
          i=i+1;

        }
  return(z);
}



int tt(int p)
{int  az,z,aa,a,at,t,ay,y,ax,x,aw,w,av,v,au,u,i=0;
  z=0;a=12;t=6;y=4;x=0;w=1;v=0;u=0;z=0;

  i=0;
  while (i< p)
        {
          printf("z=%d  ; v=%d  ; t=%d  ; w= %d; a = %d; y = %d; x = %d; u = %d;\n
          z=z+u+v+w;
          v=v+t;
          t=t+12;
          w=w+4;
          u=u+x+y;
          x=x+a;
          a=a+24;
          y=y+12;
          i=i+1;
```

```
        }
  return(z);
}


int main()
{
  int v;

  printf("Entrez la valeur 8 pour  v\n");
  scanf("%d", &v);
  printf(" voici la rÃ©ponse de votre solution %d --> %d\n",v,tt(v));
  return 0;
}
```

9 Groupe 9

9.1 Sorting

```
// Bucket sort in C

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

#define NARRAY 7   // Array size
#define NBUCKET 6  // Number of buckets
#define INTERVAL 10  // Each bucket capacity

struct Node {
  int data;
  struct Node *next;
};

void BucketSort(int arr[]);
struct Node *InsertionSort(struct Node *list);
void print(int arr[]);
void printBuckets(struct Node *list);
int getBucketIndex(int value);

// Sorting function
void BucketSort(int arr[]) {
  int i, j;
  struct Node **buckets;

  // Create buckets and allocate memory size
  buckets = (struct Node **)malloc(sizeof(struct Node *) * NBUCKET);

  // Initialize empty buckets
  for (i = 0; i < NBUCKET; ++i) {
    buckets[i] = NULL;
  }

  // Fill the buckets with respective elements
```

```
  for (i = 0; i < NARRAY; ++i) {
    struct Node *current;
    int pos = getBucketIndex(arr[i]);
    current = (struct Node *)malloc(sizeof(struct Node));
    current->data = arr[i];
    current->next = buckets[pos];
    buckets[pos] = current;
  }

  // Print the buckets along with their elements
  for (i = 0; i < NBUCKET; i++) {
    printf("Bucket[%d]: ", i);
    printBuckets(buckets[i]);
    printf("\n");
  }

  // Sort the elements of each bucket
  for (i = 0; i < NBUCKET; ++i) {
    buckets[i] = InsertionSort(buckets[i]);
  }

  printf("------------\n");
  printf("Bucktets after sorting\n");
  for (i = 0; i < NBUCKET; i++) {
    printf("Bucket[%d]: ", i);
    printBuckets(buckets[i]);
    printf("\n");
  }

  // Put sorted elements on arr
  for (j = 0, i = 0; i < NBUCKET; ++i) {
    struct Node *node;
    node = buckets[i];
    while (node) {
      arr[j++] = node->data;
      node = node->next;
    }
  }

  return;
}

// Function to sort the elements of each bucket
struct Node *InsertionSort(struct Node *list) {
  struct Node *k, *nodeList;
  if (list == 0 || list->next == 0) {
    return list;
  }

  nodeList = list;
  k = list->next;
  nodeList->next = 0;
  while (k != 0) {
    struct Node *ptr;
    if (nodeList->data > k->data) {
      struct Node *tmp;
```

```
      tmp = k;
      k = k->next;
      tmp->next = nodeList;
      nodeList = tmp;
      continue;
    }

    for (ptr = nodeList; ptr->next != 0; ptr = ptr->next) {
      if (ptr->next->data > k->data)
        break;
    }

    if (ptr->next != 0) {
      struct Node *tmp;
      tmp = k;
      k = k->next;
      tmp->next = ptr->next;
      ptr->next = tmp;
      continue;
    } else {
      ptr->next = k;
      k = k->next;
      ptr->next->next = 0;
      continue;
    }
  }
  return nodeList;
}

int getBucketIndex(int value) {
  return value / INTERVAL;
}

void print(int ar[]) {
  int i;
  for (i = 0; i < NARRAY; ++i) {
    printf("%d ", ar[i]);
  }
  printf("\n");
}

// Print buckets
void printBuckets(struct Node *list) {
  struct Node *cur = list;
  while (cur) {
    printf("%d ", cur->data);
    cur = cur->next;
  }
}

// Driver code
int main(void) {
  int array[NARRAY] = {42, 32, 33, 52, 37, 47, 51};

  printf("Initial array: ");
  print(array);
```

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

  BucketSort(array);
  printf("-------------\n");
  printf("Sorted array: ");
  print(array);
  return 0;
}
```

9.2 Power6bis.c

```
#include <stdio.h>

int p3(int x)
{int  az,z,v,av,w,aw,t,at,u=0,au;
  z=0;v=0;w=1;t=3; aw=w;az=z;at=t;av=v;au=u;
  while (u< x)
        {
          printf("z=%d  ; v=%d  ; w=%d  ; t = %d; u = %d\n",z,v,w,t,u);
          au=u;
          z=az+av+aw;
          v=av+at;
          t=at+6;
          w=aw+3;
          aw=w;az=z;at=t;av=v;
          u=u+1;

        }
  return(z);
}


int p2(int x)
{int  r,k,cv,cw,or,ok,ocv,ocw;
  r=0;k=0;cv=0;cw=0;or=0;ok=k;ocv=cv;ocw=cw;
  while (k<x)
        {
          ok=k;ocv=cv;ocw=cw;
          k=ok+1;
          cv=ocv+ocw+1;
          cw=ocw+2;}
  r=cv;return(r);
}



int p23(int x)
{int r1,r;
  r1 = p2(x);
  r = p3(r1);
    return(r);}



int main()
{
```

```
  int v;

  printf("Entrez la valeur 8 pour  v\n");
  scanf("%d", &v);
  printf(" voici la rÃ©ponse de votre solution %d --> %d\n",v,p23(v));
  return 0;
}
```

10 Groupe 10

10.1 Quick Sort

```
// Quick sort in C

#include <stdio.h>

// function to swap elements
void swap(int *a, int *b) {
  int t = *a;
  *a = *b;
  *b = t;
}

// function to find the partition position
int partition(int array[], int low, int high) {

  // select the rightmost element as pivot
  int pivot = array[high];

  // pointer for greater element
  int i = (low - 1);

  // traverse each element of the array
  // compare them with the pivot
  for (int j = low; j < high; j++) {
    if (array[j] <= pivot) {

      // if element smaller than pivot is found
      // swap it with the greater element pointed by i
      i++;

      // swap element at i with element at j
      swap(&array[i], &array[j]);
    }
  }

  // swap the pivot element with the greater element at i
  swap(&array[i + 1], &array[high]);

  // return the partition point
  return (i + 1);
}

void quickSort(int array[], int low, int high) {
  if (low < high) {
```

```
    // find the pivot element such that
    // elements smaller than pivot are on left of pivot
    // elements greater than pivot are on right of pivot
    int pi = partition(array, low, high);

    // recursive call on the left of pivot
    quickSort(array, low, pi - 1);

    // recursive call on the right of pivot
    quickSort(array, pi + 1, high);
  }
}

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

// main function
int main() {
  int data[] = {8, 7, 2, 1, 0, 9, 6};

  int n = sizeof(data) / sizeof(data[0]);

  printf("Unsorted Array\n");
  printArray(data, n);

  // perform quicksort on data
  quickSort(data, 0, n - 1);

  printf("Sorted array in ascending order: \n");
  printArray(data, n);
}
```

10.2 Power6.c

```
#include <stdio.h>

int p6(int x)

{int u=0,e=1,da=15,ca=20,cba=60,ba=15,bba=90,bca=60,bcba=180,aa=6,d=0,c=0,cb=0,b=0,

  while (u< x)
        {
          z=z+a+b+c+d+e;
          a=a+ad+bc+cb+aa;
          ad=b+b + bc+bc + bb+bb + ba+ba;
          b=b+bc+bb+ba;
          bc=bc+bcb+bca;
          bcb=bcb+bcba;
          bb=bb+bba;
          c=c+cb+ca;
          cb=cb+cba;
```

```
            d=d+da;
            aa = aa + 30;
            bcba = bcba + 360;
            bca = bca + 180;
            bba = bba + 180;
            ba = ba + 60;
            ca = ca + 60;
            da = da + 30;
            e = e + 6;
            cba = cba + 120;
            u=u+1;




        }
  return(z);
}


int fpower5(int x)

{int a=0,b=0,c=0,d=0,e=1,f=0,g=0,h=5,i=0, j=10,k=10,l=0,m=20,o=30,p=60,z=0,aa,ab,ac

  aa=a;ab=b;ac=c;ad=d;ae=e;af=f;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;ao=o;ap=p;

  while (z< x)
        {
  aa=a;ab=b;ac=c;ad=d;ae=e;af=f;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;ao=o;ap=p;
           a=a+b+c+d+e;
           b=b+f+g+h;
           c=c+i +j;
           d=d+k;
           e = e + 5;
           f=f+l+m;
           g=g+o;
           i=i+o;
           h=h+20;
           j=j+30;
           k=k+20;
           l=l+p;
           m=m+60;
           o=o+60;
           p=p+120;
           z=z+1;
           //       printf("|  %d | %d | %d | %d | %d | %d | %d | %d | %d | %d |
%d\n",a,b,c,d,e,f,g,i,k,h,j,l,m,o,p,z);
  aa=a;ab=b;ac=c;ad=d;ae=e;af=f;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;ao=o;ap=p;
        }
  return(a);
}

int main()
{
  int v;
```

```
  printf("Entrez la valeur  pour  v\n");
  scanf("%d", &v);
  // printf(" which is %d --> for power 6 %d\n",v,p6(v));
  printf("%d -->  p6=%d et p6*v %d\n",v,p6(v),fpower5(v)*v);
  return 0;
}
```

11 Groupe 11

11.1 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++)
C    {
      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");
}
```

11.2 GCD SCM

```
#include <limits.h>
#include  "structure.h"
#include  "gcdscm.h"

struct paire codegcdscm(int x1,int x2) {
  int y1 = x1;
  int y2 = x2;
  int y3 = x2;
  int y4 = 0;
  struct paire r;
  while (y1 != y2) {
    while (y1 > y2) { y1 = y1 -y2; y4=y4+y3;};
    while (y1 < y2) { y2 = y2 -y1;y3=y3+y4;};
  };
  r.gcd = y1;
  r.scm = y3+y4;
  r.wit= x1*x2;
   return r;
}

#include <stdio.h>
#include "structure.h"
#include "gcdscm.h"

int main () {

  int counter;
    for( counter=0; counter<5; counter++ ) {
      int x1,x2;
      struct paire r;
      printf("Enter a natural number x1:");
      scanf("%d", &x1);
      printf("Enter a natural number x2:");
      scanf("%d", &x2);
      r = codegcdscm(x1,x2);
      printf ("Result: %d,%d ----> %d,%d\n", x1,x2,r.gcd,r.scm);
```

```
    };
}

#include <stdio.h>




struct paire {
    unsigned gcd;
    unsigned scm;
  unsigned wit; };

struct paire codegcdscm(int x1,int x2) {
  int y1 = x1;
  int y2 = x2;
  int y3 = x2;
  int y4 = 0;
  struct paire r;
  while (y1 != y2) {
    while (y1 > y2) { y1 = y1 -y2; y4=y4+y3;};
    while (y1 < y2) { y2 = y2 -y1;y3=y3+y4;};
  };
  r.gcd = y1;
  r.scm = y3+y4;
  r.wit= x1*x2;
   return r;
}

int main () {

  int counter;
    for( counter=0; counter<5; counter++ ) {
      int x1,x2;
      struct paire r;
      printf("Enter a natural number x1:");
      scanf("%d", &x1);
      printf("Enter a natural number x2:");
      scanf("%d", &x2);
      r = codegcdscm(x1,x2);
      printf ("Result: %d,%d ----> %d,%d\n", x1,x2,r.gcd,r.scm);

    };
}
```

12 Groupe 12

12.1 Rotation d'un tableau

```
// C++ program to illustrate how to rotate array
#include <stdio.h>

// Function to rotate array left by one position
void leftRotateByOne(int arr[], int n)
{
```

```
    int temp = arr[0];
    for (int i = 0; i < n - 1; i++) {
        arr[i] = arr[i + 1];
    }
    arr[n - 1] = temp;
}

// Function to rotate array left by d positions
void leftRotate(int arr[], int d, int n)
{
    for (int i = 0; i < d; i++) {
        leftRotateByOne(arr, n);
    }
}

// Function to print an array
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++) {
        printf("%d␣", arr[i]);
    }
    printf("\n");
}

int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int d = 2;

    // Rotate array left by d positions
    leftRotate(arr, d, n);

    printf("Array␣␣after␣rotated␣by␣%d␣positions␣is:␣", d);
    printArray(arr, n);

    return 0;
}
```

12.2 Calcul du min et du max

```
/* structure is used to return two values from minMax() */
#include<stdio.h>
struct pair
{
int min;
int max;
};

struct pair getMinMax(int arr[], int n)
{
struct pair minmax;
int i;

/*If there is only one element then return it as min and max both*/
if (n == 1)
```

```
{
    minmax.max = arr[0];
    minmax.min = arr[0];
    return minmax;
}

/* If there are more than one elements, then initialize min
    and max*/
if (arr[0] > arr[1])
{
    minmax.max = arr[0];
    minmax.min = arr[1];
}
else
{
    minmax.max = arr[1];
    minmax.min = arr[0];
}

for (i = 2; i<n; i++)
{
    if (arr[i] > minmax.max)
    minmax.max = arr[i];

    else if (arr[i] < minmax.min)
    minmax.min = arr[i];
}

return minmax;
}

/* Driver program to test above function */
int main()
{
  int arr[] = {1000, 11, 445, 1, 330,-1, 3000};
int arr_size = 6;
struct pair minmax = getMinMax (arr, arr_size);
printf("nMinimum␣element␣is␣%d␣and" , minmax.min);
printf("nMaximum␣element␣is␣%d\n", minmax.max);
getchar();
}
```

13 Groupe 13

13.1 Calcul des Ã©lÃ©ments communs

```
// C Program to demonstrate scrunity of
// 2 Common Array Elements Using Merge
// Sort and then Traversing
#include <stdio.h>

int main()
{
    int array1[] = { 1, 2, 2, 3, 5, 6, 7, 8, 18, 29, 37 };
    int array2[] = { 2, 2, 4, 5, 7, 9, 10, 18 };
    int i = 0, j = 0, flag, x, k = 0;
```

```
    int result[100];

    // Calculate size of arrays
    int array1_size = sizeof(array1) / sizeof(array1[0]);
    int array2_size = sizeof(array2) / sizeof(array2[0]);

    printf("Common elements are: ");

    // Step 1
    while (i < array1_size && j < array2_size) {

        // Step 2
        if (array1[i] < array2[j]) {
            i++;
        }
        // Step 3
        else if (array1[i] > array2[j]) {
            j++;
        }
        // Step 4
        else {
            flag = 0;

            // To traverse in result array.
            for (x = 0; x < k; x++) {

                // Check whether found element is already
                // present in result array or not.
                if (result[x] == array1[i]) {
                    flag++;
                }
            }

            // If we found a new element which is common in
            // both arrays then store it in result array and
            // print it.
            if (flag == 0) {
                printf("%d ", array1[i]);
                result[k] = array1[i];
                k++;
            }
            i++;
            j++;
        }
    }
}
```

13.2 Fusipn de deux tableaux

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

// This function is to check
// if a given number is prime
bool isPrime(int n)
```

```
{
    // 0 and 1 are not prime numbers
    if (n == 1 || n == 0)
        return false;

    // Check for divisibility from 2 to sqrt(n)
    for (int i = 2; i <= sqrt(n); i++) {
        if (n % i == 0)
            return false;
    }
    return true;
}

// Driver code
int main()
{
    int N = 50;

    // Check every number from 1 to N
    for (int i = 1; i <= N; i++) {
        if (isPrime(i)) {
            printf("%d ", i);
        }
    }

    return 0;
}
```

14 Groupe 14

14.1 Calcul des Ã©lÃ©ments communs

```
// C Program to demonstrate scrunity of
// 2 Common Array Elements Using Brute force
#include <stdio.h>

int main()
{
    int array1[] = { 8, 2, 3, 4, 5, 6, 7, 1 };
    int array2[] = { 4, 5, 7, 11, 6, 1 };
    int i, j, flag, x, k = 0;
    int result[100];
    printf("Common elements are: ");

    // To traverse in array1.
    for (i = 0; i < sizeof(array1) / 4; i++) {

        // To traverse in array2.
        for (j = 0; j < sizeof(array2) / 4; j++) {

            // To match elements of array1 with elements of
            // array2.
            if (array1[i] == array2[j]) {

                flag = 0;
```

```
                // To traverse in result array.
                for (x = 0; x < k; x++) {

                    // Check whether found element is
                    // already present in result array or
                    // not.
                    if (result[x] == array1[i]) {
                        flag++;
                    }
                }

                // If we found a new element which is common
                // in both arrays then store it in result
                // array and print it.
                if (flag == 0) {

                    result[k] = array1[i];
                    printf("%d ", result[k]);
                    k++;
                }
            }
        }
    }
}
```

14.2 Fusipn de deux tableaux

```
// C program to merge two arrays into a new array using
// memcpy()
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int* mergeArrays(int arr1[], int n1, int arr2[], int n2) {

        // Resultant array to store merged array
        int *res = (int*)malloc(sizeof(int) * n1 * n2);

    // Copy elements of the first array
    memcpy(res, arr1, n1 * sizeof(int));

    // Copy elements of the second array
    memcpy(res + n1, arr2, n2 * sizeof(int));

        return res;
}

int main() {
    int arr1[] = {1, 3, 5};
    int arr2[] = {2, 4, 6};
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    int n2 = sizeof(arr2) / sizeof(arr2[0]);

        // Merge arr1 and arr2
    int* res = mergeArrays(arr1, n1, arr2, n2);
```

```
    for (int i = 0; i < n1 + n2; i++)
        printf("%d ", res[i]);

    return 0;
}
```

15 Groupe 15

15.1 Suppression d'un Ã©lÃ©ment

```
// C Program to remove All Occurrences of
// an element in an array using functions
#include <stdio.h>

// function declaration
int remove_element(int* array, int n, int val)
{
    int i;

    // iterates array elements
    for (i = 0; i < n; i++)

        // if the given array element is not equal to given
        // value then print array element
        if (array[i] != val)
            printf("%d ", array[i]);
}

int main()
{
    int array[6] = { 1, 2, 1, 3, 1 }, size = 5, value = 1;
    remove_element(array, size, value);
}
```

15.2 VÃ©rification de l'Ã©galitÃ© de deux matrices

```
// C Program to check if two matrices are equals or not
#include <stdio.h>
#define N 4 // Macros

// This function returns 1 if A[][] and B[][] are equal
// otherwise it returns 0
int areSame(int A[][N], int B[][N])
{
    int i, j;
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            if (A[i][j] != B[i][j])
                return 0;
    return 1;
}

// driver code
int main()
{
```

```
    // create two matrices
    int A[N][N] = { { 1, 2, 3, 4 },
                    { 1, 2, 3, 4 },
                    { 1, 2, 3, 4 },
                    { 1, 2, 3, 4 } };

    int B[N][N] = { { 1, 2, 3, 4 },
                    { 1, 2, 3, 4 },
                    { 1, 2, 3, 4 },
                    { 1, 2, 3, 4 } };

    // Display first matrix
    printf("\n First Matrix \n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf(" %d", A[i][j]);
        }
        printf("\n");
    }

    // Display second matrix
    printf("\n Second Matrix \n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf(" %d", B[i][j]);
        }
        printf("\n");
    }

    // check if Matrices are equals or not
    // areSame(A, B) function accept two matrices as input
    // and return 1 if matrices are equals otherwise return
    // 0
    if (areSame(A, B))
        printf("\n Matrices are equal");
    else
        printf("\n Matrices are not equal");
    return 0;
}
```

16 Groupe 16

16.1 Destruction des duplicas

```
#include <stdio.h>

int removeDup(int arr[], int n) {
    if (n == 0) return 0;

    int j = 0;
    for (int i = 1; i < n; i++) {

        // If a unique element is found, place
        // it at arr[j + 1]
        if (arr[i] != arr[j])
            arr[++j] = arr[i];
```

```
    }

        // Return the new ending of arr that only
        // contains unique elements
    return j + 1;
}

int main() {
    int arr[] = {1, 1, 2, 2, 3, 4, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Remove duplicates
    n = removeDup(arr, n);

    for (int i = 0; i < n; i++)
        printf("%d␣", arr[i]);
    return 0;
}
```

16.2

```
#include <stdio.h>

float getAvg(int arr[], int n) {
    int sum = 0;

    // Find the sum of all elements
    for (int i = 0; i < n; i++) {
        sum += arr[i];
    }

        // Return the average
    return (float)sum / n;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

        // Calculate the average of array arr
    float res = getAvg(arr, n);

    printf("%.2f\n", res);
    return 0;
}
```