SCHRECK
Corentin
MP2I

DM n°2

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

//exo 20
int spitsize(bool tab[],int size)
{
  int nb=1;
  for (int i=0; i<size; i+=1)
  {
    if (tab[i]==tab[0])
    {
      nb+=1;
    }
    else return nb;
  }
}

//exo 21
int lasttrue(bool tab[],int size)
{
  int i=size-1;
  while (!tab[i] && i<0)
  {
    i-=1;
  }
  return i;
}
bool nul(bool tab[], int size)
{
  int t=lasttrue(tab,size);
  int nbtrue=0;
  int nbfalse=0;
  for (int i=0;i<t;i+=1)
  {
    if (tab[i]) nbtrue+=1;
    else nbfalse+=1;
    if (nbtrue<nbfalse) return false;
  }
  return true;

}

//exo 22
double modfloat(double a, double b)
{
  if (a==0)
  {
    return a;
  }
  double t=-1.0;
  if (a*b<0)
  {
    t=1.0;
  }
  if (a>0)
  {
    while(a>=abs(b))
    {
      a+=t*b;
    }
  }
  else
  {
    while(a<=abs(b))
    {
      a-=t*b;
```

SCHRECK
Corentin
MP2I

```
    }
  }
  return a;

}

//exo 23
int *minmax(int tab[], int size)
{
  int min=tab[0];
  int max=min;
  for (int i=0; i<size ; i+=1)
  {
    if (tab[i]<min) min=tab[i];
    if (tab[i]>max) max=tab[i];
  }
  int *m=malloc(2*sizeof(int));
  m[0]=min;
  m[1]=max;
  return m;
}

//exo 24
int medianemax(int tab[], int size)
{
  int max=tab[0];
  int nbmax=1;
  int* maxtab=malloc(size*sizeof(int));
  maxtab[0]=0;
  for (int i=1;i<size;i+=1)
  {
    if (tab[i]==max)
    {
      maxtab[nbmax]=i;
      nbmax+=1;
    }
    if (tab[i]>max)
    {
      max=tab[i];
      nbmax=1;
      maxtab[0]=i;
    }
  }
  return maxtab[((nbmax+1)/2)-1];

}

//exo 25
double* derive(double tab[],int size)
{
  double* d=malloc(size*sizeof(double));
  for (int i=0;i<size-1;i+=1)
  {
    d[i]=tab[i+1]*(i+1);
  }
  d[size-1]=0;
  return d;
}

//exo 26
double* multpol(double tab1[], double tab2[], int size1, int size2)
{
  int poldeg=(size1-1)+(size2-1);
  double* m=malloc((poldeg+1)*sizeof(double));
  for (int i=0;i<=poldeg;i+=1)
  {
    m[i]=0;
  }
  for (int i=0;i<size1;i+=1)
  {
```

SCHRECK
Corentin
MP2I

```
    for (int ii=0;ii<size2;ii+=1)
    {
      m[i+ii]+=tab1[i]*tab2[ii];
    }
  }
  return m;
}

//exo 27
bool in(int e, int tab[], int size)
{
  for (int i=0;i<size;i+=1)
  {
    if (e==tab[i])
    {
      return true;
    }
  }
  return false;
}
char* cesar(char string[], int a, int b)
{
  //vérification de la bijectivité de l'encodage
  int results[26];
  for (int i=1; i<=26; i+=1)
  {
    int code=(a*i+b)%26;
    if (code<=0) code+=26;
    if(in(code,results,i-1))
    {
      //déclenche une erreur car renvoie le mauvais type
      return 1;
    }
    else results[i]=code;
  }

  //encodage du message
  int l=(int)strlen(string);
  char* msg=malloc((l+1)*sizeof(char));
  for (int i=0;i<=l;i+=1)
  {
    int c=(int)string[i];
    if((int)'a'<=c && c<=(int)'z')
    {
      c-=(int)'a'-1;
      c=(a*c+b)%26;
      if (c<=0)
      {
        c+=26;
      }
      c+=(int)'a'-1;
    }

    if((int)'A'<=c && c<=(int)'Z')
    {
      c-=(int)'A'-1;
      c=(a*c+b)%26;
      if (c<=0)
      {
        c+=26;
      }
      c+=(int)'A'-1;
    }
    msg[i]=(char)c;
  }
  return msg;
}

//exo 28
char* auguste(int nb)
```

SCHRECK
Corentin
MP2I

```
{
  if (!(0<=nb && nb<=3999)) return 1; //déclenche une erreur car renvoie le mauvais type
  char* nbrom=malloc(16*sizeof(char));
  int i=0;
  while (nb>=1000)
  {
    nb-=1000;
    nbrom[i]='M';
    i+=1;
  }
  if (nb>=500)
  {
    nb-=500;
    nbrom[i]='D';
    i+=1;
  }
  if (nb>=400)
  {
    nb-=400;
    nbrom[i]='C';
    nbrom[i+1]='D';
    i+=2;
  }
  while (nb>=100)
  {
    nb-=100;
    nbrom[i]='C';
    i+=1;
  }
  if (nb>=50)
  {
    nb-=50;
    nbrom[i]='L';
    i+=1;
  }
  if (nb>=40)
  {
    nb-=40;
    nbrom[i]='X';
    nbrom[i+1]='L';
    i+=2;
  }
  while (nb>=10)
  {
    nb-=10;
    nbrom[i]='X';
    i+=1;
  }
  if (nb>=5)
  {
    nb-=5;
    nbrom[i]='V';
    i+=1;
  }
  if (nb>=4)
  {
    nb-=4;
    nbrom[i]='I';
    nbrom[i+1]='V';
    i+=2;
  }
  while (nb>=1)
  {
    nb-=1;
    nbrom[i]='I';
    i+=1;
  }
  nbrom[i]='\0';
  return nbrom;
}
```

SCHRECK
Corentin
MP2I

```
//exo 29
int* ecart(int tab[], int size)
{
  int* elements=malloc(size*sizeof(int));
  int* occurences=malloc(size*sizeof(int));
  occurences[0]=1;
  elements[0]=tab[0];
  int nbe=0;
  for (int i=1;i<size;i+=1)
  {
    bool test=false;
    for(int j=0;j<=nbe;j+=1)
    {
      if (tab[i]==elements[j])
      {
        occurences[j]+=1;
        test=true;
        break;
      }
    }
    if (!test)
    {
      nbe+=1;
      elements[nbe]=tab[i];
      occurences[nbe]=1;
    }
  }

  int* rep=malloc(2*sizeof(int));
  int min=occurences[0];
  int imin1=0;
  for (int i=1;i<=nbe;i+=1)
  {
    if (occurences[i]<min)
    {
      min=occurences[i];
      imin1=i;
    }
  }
  if (min>1)
  {
    rep[0]=elements[imin1];
    rep[1]=elements[imin1];
  }
  else
  {
    rep[0]=elements[imin1];
    min=size+1;
    int imin2;
    for (int i=0;i<nbe;i+=1)
    {
      if (i!=imin1)
      {
        if (occurences[i]<min)
        {
          min=occurences[i];
          imin2=i;
        }
      }
    }
    rep[1]=elements[imin2];
  }
  return rep;
}

int main()
{
  printf("\n[exo 20]\n");
  bool tab1[5]={true,true,false,false,true};
```

SCHRECK
Corentin
MP2I

```
  printf("nombre d'elements consecutifs identiques en partant du premier : %d \n",spitsize(tab1,5));

  printf("\n[exo 21]\n");
  printf("il y a au moins autant de true que de false a chaque indice en partant du premier : %d\n",nul(tab1,5));
  tab1[1]=false;
  printf("il y a au moins autant de true que de false a chaque indice en partant du premier : %d\n",nul(tab1,5));

  printf("\n[exo 22]\n");
  printf("2%%1.5=%f\n",modfloat(2.0,1.5));
  printf("2%%-1.5=%f\n",modfloat(2.0,-1.5));

  printf("\n[exo 23]\n");
  int tab2[5]={3,2,8,7,12};
  int* m=minmax(tab2,5);
  printf("minimum : %d, maximum : %d\n",m[0],m[1]);
  free(m);

  printf("\n[exo 24]\n");
  int tab3[5]={5,2,5,1,5};
  printf("indice median du maximum : %d \n",medianemax(tab3,5));
  tab3[1]=5;
  printf("indice median du maximum : %d \n",medianemax(tab3,5));
  tab3[0]=0;
  printf("indice median du maximum : %d \n",medianemax(tab3,5));

  printf("\n[exo 25]\n");
  double polynome[3]={1,2,3};
  double* derivee=derive(polynome,3);
  printf("derivee : %f+%f*x+%f*x^2 \n",derivee[0],derivee[1],derivee[2]);
  free(derivee);

  printf("\n[exo 26]\n");
  double* product=multpol(polynome,polynome,3,3);
  printf("produit : %f",product[0]);
  for (int i=1;i<5;i+=1)
  {
    printf("+%f*x^%d",product[i],i);
  }
  printf("\n");
  free(product);

  printf("\n[exo 27]\n");
  char msg[12]="Hello World";
  char* code=cesar(msg,1,5);
  printf("%s -> %s \n",msg,code);
  printf("%s <- %s \n",code,cesar(code,1,-5));
  free(code);

  printf("\n[exo 28]\n");
  printf("18 : %s \n",auguste(18));
  printf("42 : %s \n",auguste(42));

  printf("\n[exo 29]\n");
  int tab4[5]={1,1,1,2,2};
  int* e=ecart(tab4,5);
  printf("elements minimisant les occurences dans tab3 : %d,%d \n",e[0],e[1]);
  tab4[4]=1;
  e=ecart(tab4,5);
  printf("elements minimisant les occurences dans tab3 : %d,%d \n",e[0],e[1]);
  tab4[0]=0;
  e=ecart(tab4,5);
  printf("elements minimisant les occurences dans tab3 : %d,%d \n",e[0],e[1]);



  return 0;
}
```