Examen “électronique programmable 2”

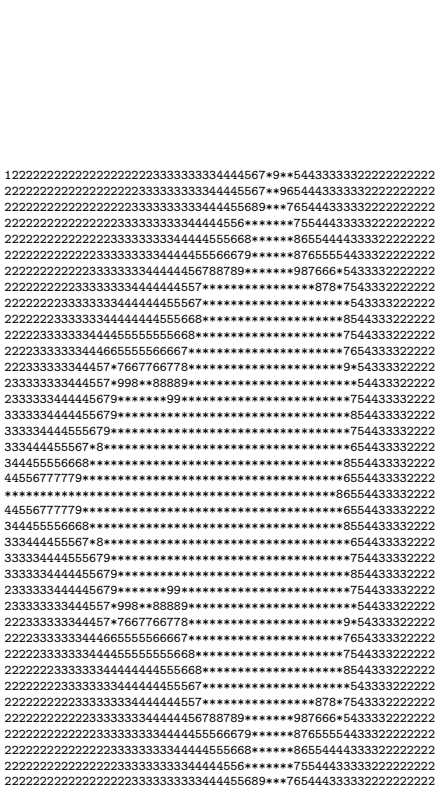
J.-M Friedt, 3 avril 2025

Nous nous proposons d'afficher l'ensemble des conditions initiales $z_0 \in \mathbb{C}$ qui vérifient que la suite $z_{n+1} = z_n^2 + z_0$ ne diverge pas. Cet ensemble se nomme la fractale de Mandelbrot et apparaît naturellement lors de l'étude de systèmes dynamiques.

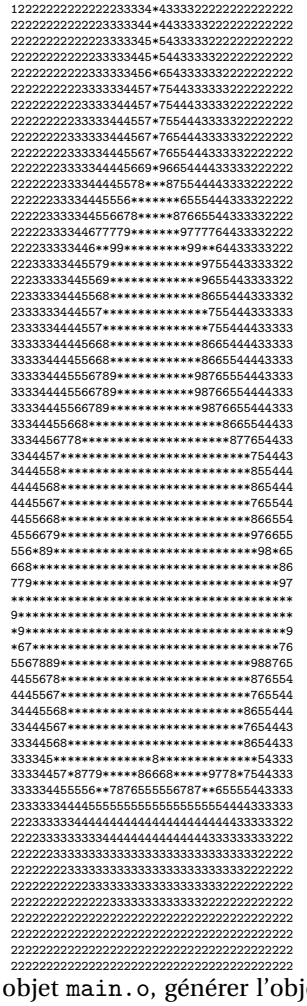
Pour effectuer les calculs sur les nombres complexes, nous proposons à `http://jmfriedt.free.fr/examEP2_2025.tar.gz` une bibliothèque implémentant en C un complexe formé d'une partie réelle et d'une partie imaginaire, ainsi que les opérations de base que sont la multiplication complexe, l'addition complexe et le calcul du carré du module du complexe. Bien que l'utilisation de la représentation des nombres à virgule flottante (`float` en C) soit proscrite sur microcontrôleur, nous nous autoriserons son utilisation pour cette démonstration compte tenu de nos connaissances actuelles.

Nous commençons par travailler sur PC et son processeur compatible Intel x86 (questions 1 à 3) avant de passer à l'Atmega32U4 :

1. proposer un programme dans un fichier nommé `main.c` qui fasse appel aux fonctions de `cplx.c` qui balaie la gamme de $-2,0$ à $+1,0$ par pas de $0,05$ selon l'axe des abscisses (partie réelle de z_0) et de $-1,0$ à $+1,0$ par pas de $0,05$ selon l'axe des ordonnées (partie imaginaire de z_0) sous forme de deux boucles imbriquées, et qui pour chaque valeur de z_0 calcule la suite $z_{n+1} = z_n^2 + z_0$. Le calcul de la suite doit cesser si
   - soit le module au carré de z_n dépasse 10
   - soit le nombre d'itérations n dépasse 10.
2. pour chaque valeur de z_0, afficher à l'écran soit la valeur de n si elle est inférieure à 10, soit “*” si la valeur de 10 a été atteinte (i.e. le module de z_n n'a pas dépassé 10, donc la suite n'a pas divergé). On ajoutera un retour à la ligne chaque fois que la partie réelle atteint sa borne supérieure de 1. Le résultat sera un graphique de la forme



ou



3. proposer un `Makefile` pour convertir `main.c` en objet `main.o`, générer l'objet contenant les fonctions de calcul sur les nombres complexes, et lier en un exécutable. On proposera aussi une méthode `nettoie` dans le `Makefile` qui efface de façon inconditionnelle les objets et l'exécutable générés lors de la compilation.

4. La fonction `printf()` ne peut être utilisée sur Atmega32U4 puisque la bibliothèque qui implémente cette fonction ne peut savoir vers quel périphérique diriger l'affichage. Modifier le programme afin de rendre la compilation des instructions `printf()` conditionnelle à la compilation sur PC et exclure ces fonctions si la compilation se fait vers une cible Atmega32U4. On notera que le compilateur `gcc` définit la constante `__amd64__` lors de la compilation sur processeur compatible Intel, ou `__aarch64__` sur les ordinateurs Apple basés sur processeur ARM.
5. Modifier le programme afin de faire appel aux fonctions d'initialisation du bus USB et d'affichage de la bibliothèque LUFA si la cible de la compilation n'est *pas* un processeur Intel, *i.e.* si le compilateur utilisé est `avr-gcc`.
6. Proposer un nouveau `Makefile` qui automatise la compilation du programme selon la même séquence que précédemment, mais cette fois à destination de l'Atmega32U4. On se rappelle que si le fichier de configuration de `make` ne se nomme pas `Makefile` ou `makefile` mais `monmake`, alors son utilisation s'obtient par `make -f monmake`.
7. Démontrer le bon fonctionnement de votre programme exécuté sur Atmega32U4. Comment affichez vous le résultat du calcul par l'Atmega32U4?
8. Combien de temps nécessite le calcul et l'affichage de la fractale de Mandelbrot sur Atmega32U4? On pourra pour ce faire s'appuyer sur une mesure à l'oscilloscope en observant un signal adéquat produit par le microcontrôleur en début et en fin de calcul.
9. Convaincus que le calcul s'effectue convenablement, commenter les fonctions de communication sur bus USB et reproduire la mesure précédente qui cette fois n'inclut que le temps de calcul et n'inclut *plus* le temps de communication. Comment se comparent les deux valeurs, temps de calcul et de communication d'une part et temps de calcul uniquement?
10. Combien d'octets en mémoire non-volatile (*flash*) sont occupés par le programme? Comment cette valeur se compare au volume totale de mémoire non-volatile disponible sur l'Atmega32U4?

Solution :

1. Le premier programme vise à démontrer que le fichier d'entête fourni

```
#include "math.h"

struct cpl {float re; float im;};

struct cpl cpl_mul(struct cpl , struct cpl );
struct cpl cpl_add(struct cpl , struct cpl );
float cpl_mod2(struct cpl );
```

est inclus pour la compilation séparée de la bibliothèque fournie

```
#include "cplx.h"

struct cpl cpl_mul(struct cpl x1, struct cpl x2)
{struct cpl res;
 res.re=x1.re*x2.re-x1.im*x2.im;
 res.im=x1.re*x2.im+x1.im*x2.re;
 return(res);
}

struct cpl cpl_add(struct cpl x1, struct cpl x2)
{struct cpl res;
 res.re=x1.re+x2.re;
 res.im=x1.im+x2.im;
 return(res);
}

float cpl_mod2(struct cpl x1)
{return(x1.re*x1.re+x1.im*x1.im);
}
```

et que la structure de données contenant les complexes est correctement déclarée et les fonctions de calcul sur les complexes convenablement appelées dans les deux boucles imbriquées pour le calcul de la suite. Le résultat complet est fourni ci-dessous...

2. ... avec le programme `main.c` qui trace la fractale de Mandelbrot sur PC de la forme

```
#include "cplx.h"

#ifdef __amd64__
#include <stdio.h>
#endif

int main()
{struct cpl c,c0;
 float x,y;
 int n;
 for (y=-1.;y<1.;y+=0.05)
   {for (x=-2;x<1.;x+=0.05)
     {n=0;
      c.re=x;c.im=y;
      c0.re=x;c0.im=y;
      do {c=cpl_add(cpl_mul(c,c),c0);
          n++;
         } while ((n<10)&&(cpl_mod2(c)<10.));
#ifdef __amd64__
      if (n==10) printf("*"); else printf("%d",n);
#endif
     }
#ifdef __amd64__
   printf("\n");
#endif
   }
}
```

3. La compilation s'obtient par le `Makefile` suivant :

```
all: main

main: main.o cplx.o
   gcc -o main main.o cplx.o

cplx.o: cplx.c
   gcc -c cplx.c

main.o: main.c
   gcc -c main.c

nettoie:
   rm main *.o
```

4. La compilation conditionnelle par le préprocesseur encapsule les `printf` dans `#ifdef __amd64__`...
5. et le programme `main.c` pour Atmega32U4 qui appelle les fonctions de LUFA pour initialiser le port USB et remplit le tampon par `fputc` est

```
#include "cplx.h"

#ifdef __amd64__
#include <stdio.h>
#endif

int main()
{struct cpl c,c0;
 float x,y;
 int n;
 for (y=-1.;y<1.;y+=0.05)
   {for (x=-2;x<1.;x+=0.05)
     {n=0;
      c.re=x;c.im=y;
      c0.re=x;c0.im=y;
      do {c=cpl_add(cpl_mul(c,c),c0);
          n++;
         } while ((n<10)&&(cpl_mod2(c)<10.));
#ifdef __amd64__
      if (n==10) printf("*"); else printf("%d",n);
#endif
     }
#ifdef __amd64__
   printf("\n");
#endif
   }
}
```

Ce programme peut se compiler à la fois pour le PC et l'Atmega par la sélection des bouts de codes inclus par le préprocesseur en fonction du compilateur sélectionné.

6. Le `Makefile` pour Atmega32U4 est de la forme

```
REP=/home/jmfriedt/enseignement/sqnx/platforms/Atmega32

CFLAGS=-Wall -I$(REP)/VirtualSerial/ -I$(REP)/lufa-LUFA-140928 \
       -DF_USB=16000000UL -DF_CPU=16000000UL -Os -std=gnu99

CPU=atmega32u4

all: main.avr

main.avr: main.o cplx.o
   avr-gcc -mmcu=$(CPU) -o main.avr main.o cplx.o -L$(REP)/VirtualSerial -lVirtualSerial
   avr-objcopy -O binary main.avr main.bin
   avr-objcopy -O ihex main.avr main.hex
```

```
14
15 cplx.o: cplx.c
16    avr-gcc -mmcu=$(CPU) $(CFLAGS) -c cplx.c
17
18 main.o: main.c
19    avr-gcc -mmcu=$(CPU) $(CFLAGS) -c main.c
20
21 nettoie:
22    rm main main.bin *.o
23
24    avr-objdump -dSt $(TARGET).out > $(TARGET).lst
25    avr-objcopy -O ihex $(TARGET).out $(TARGET).hex
26
27 flash: main.avr
28    avrdude -c avr109 -b57600 -D -p $(CPU) -P /dev/ttyACM0 -e -U flash:w:main.bin
29
30 DFU=dfu-programmer
31 flash_dfu: main.avr
32    $(DFU) $(CPU) erase
33    $(DFU) $(CPU) flash main.hex
34    $(DFU) $(CPU) reset
```

7. Le résultat est strictement identique lors de l'exécution sur Atmega32U4 et affichage dans `minicom` par exemple, et sur PC.
8. Commuter l'état d'un GPIO en début et fin de boucle et mesurer sur oscilloscope le temps d'exécution. Avec la communication, le temps d'exécution mesuré à l'oscilloscope est 1,72 s.
9. Commuter l'état d'un GPIO en début et fin de boucle et mesurer sur oscilloscope le temps d'exécution. Sans communication, le temps d'exécution mesuré à l'oscilloscope est 1,70 s. La communication USB est très rapide et l'émulation des nombres à virgule flottante (`float`) très lent sur Atmega32U4. La majorité du temps est passée sur le calcul et non sur la communication dans ce cas.
10. `avr-objcopy -O binary main main.bin` pour avoir la taille du binaire (la taille ne peut pas se déduire du fichier au format ELF). Nous constatons que la taille du binaire est 9460 octets, bien plus petit que la taille de la mémoire non-volatile disponible.