Projet Architecture des Ordinateurs

-

Statistique Descriptive

YANG Liyun JIN Qianhui

- 2019 2020 -

Préambule

Notre projet en Architecture des Ordinateurs est: “Statistique descriptive”. Nous avons en processions une série des note de 150 élèves à un examen. À partir de ces notes, les programmes en assembleur de “Statique Descriptive” génèrent un histogramme ou une courbe qui permet de représenter les fréquences des notes obtenus. C’est à dire qu’ils peuvent d’indiquer pour chaque notes, le nombre d'élèves qui l'ont eu sous forme d'histogramme.

Sommaire

I. Analyse de projet

Pour afficher un histogramme à partir des notes obtenues d'un examen dans le fichier, il faut d'abord faire une fonction pour lire ce fichier. Puis il nécessite d'avoir une fonction qui permettra de compter le nombre d'élèves ayant eu la même note. Et pour finir, une fonction affichage doit être générée pour afficher un histogramme en mode texte.

À ce propos, nous avons pensé à représenter les fréquences des notes obtenus par des intervalles de notes, avec un écart de 1. Par exemple la fréquence des notes sur [1, 2[, [2, 3[, [5, 6[. Cela a besoin beaucoup de comparaisons. Après avoir discuté avec M. Marchetti, nous avons changé la façon de représentations, c'est-à-dire de compter le nombre d'élèves qui ont eu la même note. Par exemple:

*

* * *

==========

0 1 2 3 4 5 6 7.5 8 (il y a 2 élèves qui ont eu 0, 1 élève a eu 1, et 1 élève a eu 3)

Pour faciliter de compter les élèves qui ont eu la même note, nous avons décidé d'ajouter une fonction à classer les notes obtenues en ordre croissant. Cela permet d'éviter faire pleins de fois à accéder dans la boucle.

Afin de faciliter la programmation, nous avons choisi de faire le programmation en C dans un premier temps, puis on le traduit en MIPS.

II. Les programmes en C

Tout d'abord, nous avons écrit la fonction "lecture". Cette fonction permettra de lire les notes stockées dans le fichier "notes.txt". Dans cette fonction, nous ouvrons d'abord le fichier "notes.txt", ensuite nous utilisons un pointeur pour allouer un tableau de fractions décimals qui va permettre de stocker les notes, puis la fonction "fscanf" va récupérer les notes du fichier "notes.txt" dans ce tableau. Et nous n'oublions pas de fermer le fichier à la fin.

Ensuite, nous écrivons la fonction "tabNoteEleve" pour classer les notes. Nous comparons chaque note avec les autres suivantes, s'il y a d'une note qui est plus petite que la note comparée, alors nous changeons leurs positions. Finalement nous allons obtenir un tableau de notes en ordre croissante.

Puis nous allons compter la fréquence des notes. Comme nous ne connaissons pas la totalité de notes qui vont apparaître, nous ne pouvons donc pas

définir la taille du tableau. Ainsi, nous pensons à créer une nouvelle structure “t_frequnce” qui contient la note et la fréquence de notes, en plus un pointeur vers l’élément suivant. Puis nous écrivons une fonction “liste_create” qui prend en argument d’une note en forme de la fraction décimale “f” et d’une fréquence en forme d’un nombre entier “nb” afin d’initialiser un élément de type “t_frequence” de la liste chaînée. Cette fonction récupére “f” et “nb” dans l’élément de la liste et elle retourne le pointeur de l’élément de la liste.

Ensuite, la fonction "append" permettra d'ajouter un élément à la fin de la liste chaînée en créant les éléments de la liste et en reliant la liaison entre les éléments.

Maintenant, nous passons à l’étape compteur. Faisons une fonction “listeDeFrequence” en prenant deux arguments: le tableau de note classé et la taille du tableau. Dans cette fonction, nous comptons la fréquence de chaque note, puis créons la liste chaînée en appelant la fonction “append” (donc nous appelons aussi la fonction “liste_create”). Cette fonction retournera le premier élément de la liste chaînée.

À la fin, nous faisons deux représentations différentes, le premier affiche directement le nombre d’élève pour chaque note, l’autre affiche l'histogramme en mode text.

III. Les programmes en MIPS

Après avoir fini la programmation en C, nous commençons à traduire ces fonctions de C en MIPS. Mais cela n’est pas exactement la même, il y a des différences. Donc nous décidons de simplifier certaine fonction, nous faisons juste une fonction de lecture, une fonction de classement, une fonction de compteur et une fonction d’affichage en mode text.

En début nous choisissons de faire la boucle switch pour compter chaque note, mais si nous utilisons ce moyen, les nombres de registre ne suffisent pas pour compter tous les notes. Donc nous changeons par la liste chainée, mais cela ne marche pas non plus, car nous ne savons pas comment faire la liste chaînée dans MIPS.

IV. Les difficultés rencontrées

Nous avons eu une difficulté importante sur la liste chaînée, notamment sur la liaison de chaque élément de la liste. D’abord nous avons pensé à créer deux

nouvelles structure: l'un est élément de la liste "t_frequence" qui contient la note, la fréquence et la liaison avec l'élément suivant; l'autre est la liste totale "t_liste" qui contient le début de la liste chaînée et la fin de la liste.

D'après l'exécution de la fonction, nous avons trouvé que la liaison entre chaque élément ne fonctionne pas, parce qu'il n'y a que le dernier élément dans la liste.

```c
t_liste* liste(float note[], int taille){
    int cpt = 1;
    float n = note[0];
    int change = 0;

    t_frequence* f = (t_frequence*) malloc (sizeof(t_frequence));

    t_liste* noteEleve = (t_liste*) malloc(sizeof(t_liste));

    if ((f  == NULL) ||(noteEleve == NULL)){
        exit(EXIT_FAILURE);
    }
    noteEleve -> fin = NULL;

    for (int i = 1; i < taille; i++)
    {
        if (note[i] == n){
            cpt++;
        }else{
            f -> noteEl = n;
            f -> nombre = cpt;
            n = note[i];
            cpt = 1;
            change ++ ;
            printf("\n%f %d", f->noteEl, f-> nombre );
        }

        if (change == 1){
            noteEleve -> debut = f;

        }else{
            if (i == taille-1){
                noteEleve -> fin = f;
            }
        }
    }
    return noteEleve;
}
```

(la fonction de la construction de la liste)

```
2.000000 3.000000 3.000000 4.000000 5.500000 7.000000 7.000000 8.000000 10.000000 10.500000 13.000000 13.000000 16.000000 17.000000 19.500000
2.000000 1
3.000000 2
4.000000 1
5.500000 1
7.000000 2
8.000000 1
10.000000 1
10.500000 1
13.000000 2
16.000000 1
17.000000 1
Pour 17.000000 , il y a 1 élèves.NULL
```

(l'affichage de la liste, il n'y a que le dernier élément dans la liste, ainsi la dernière note 19 n'a pas été construit comme un élément)

Alors, nous avons ajouté le dernier élément à la fin de la liste:

```c
        if (change == 1){
            noteEleve -> debut = f;
            printf("\n%f %d", f -> noteEl, f-> nombre );
```

```
2.000000 1
3.000000 2
4.000000 1
5.500000 1
7.000000 2
8.000000 1
10.000000 1
10.500000 1
13.000000 2
16.000000 1
17.000000 1
19.500000 1
```

D'après l'ajout des conditions dans la construction de la liste, elle ne contient que toujours que la dernière note (le dernier élément) :

```
t_liste* liste(float note[], int taille){

    t_liste *tableau = (t_liste*) malloc (sizeof(t_liste));
    t_frequence *f = (t_frequence*) malloc (sizeof(t_frequence));
    t_frequence *p = NULL;

    int cpt = 1;
    float n = note[0];
    int change = 0;


    if (f  == NULL){
        exit(EXIT_FAILURE);
    }

    for (int i = 1; i < taille; i++)
    {
        if (note[i] == n){
            cpt++;
        }else{
            f -> noteEl = n;
            f -> nombre = cpt;
            n = note[i];
            cpt = 1;
            change ++ ;
            printf("\n%f %d", f -> noteEl, f-> nombre );
        }

        if ((change == 1) && (cpt == 1)){
            p = f;
            tableau -> debut = p;
            printf("\n%f   %d", tableau -> debut -> noteEl, tableau -> debut -> nombre);
        }else {
            if (( change != 1) && (cpt==1)){
                p -> suivant = f;
                f = p;
                printf("\n%f   %d", p -> suivant -> noteEl, p -> suivant -> nombre);
            }
        }

        if (i == taille-1 ){
            f -> noteEl = n;
            f -> nombre = cpt;
            p -> suivant = f;
            f -> suivant = NULL;
            tableau -> fin = f;
            printf("\n%f %d", f -> noteEl, f-> nombre);
        }
    }
    return tableau;
}
```

```
2.000000 3.000000 3.000000 4.000000 5.500000 7.000000 7.000000 8.000000 10.00000
0 10.500000 13.000000 13.000000 16.000000 17.000000 19.500000
2.000000 1
2.000000   1
3.000000 2
3.000000   2
4.000000 1
4.000000   1
5.500000 1
5.500000   1
7.000000 2
7.000000   2
8.000000 1
8.000000   1
10.000000 1
10.000000   1
10.500000 1
10.500000   1
13.000000 2
13.000000   2
16.000000 1
16.000000   1
17.000000 1
17.000000   1
19.500000 1
19.500000 1
Pour 19.500000 , il y a 1 élèves.NULL
```

Dans ce cas, nous avons fait pleins de recherche, et nous avons trouvé que ce serait mieux de faire des fonctions à part: une fonction de création d'élément et une fonction à faire la liaison (l'ajout d'un élément à la fin de la liste). Puis nous ferons l'appel de ces fonctions dans la construction de la liste. En fin, ce problème a été résolu.

```
t_frequence *liste_create(float f, int nb){
    t_frequence* liste = malloc(sizeof(t_frequence));
    if (liste)
    {
        liste -> noteEl = f;
        liste -> nombreEl = nb;
        liste -> suivant = NULL;
    }
    return liste;
}
```

```
t_frequence *append(t_frequence *debut, float f, int nb){

    t_frequence *nouveau = malloc(sizeof(t_frequence));

    if (nouveau != NULL){
        nouveau = liste_create(f, nb);

        if ( debut == NULL){
            debut = nouveau;
        }
        else{
            t_frequence *inter = debut;
            while (inter -> suivant != NULL){
                inter = inter -> suivant;
            }
            inter -> suivant = nouveau;
        }
    }
    return debut;
}
```

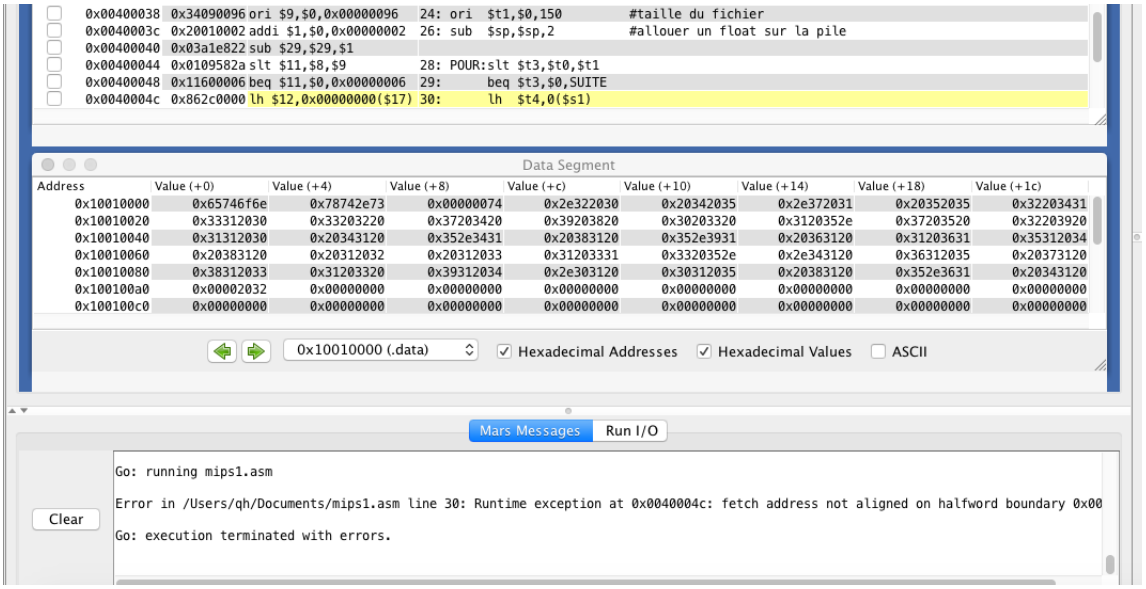
```
t_frequence *listeDeFrequence(float note[], int taille)
{
    float n = note[0];
    int cpt = 1;
    int change = 0;
    t_frequence *debutDeListe = NULL;

    for (int i = 1; i < taille; i++){

        if (note[i] == n){
            cpt ++;
        }else {
            change ++ ;
        }

        if ((change == 1) && (debutDeListe == NULL)){
            printf(" *******\n");
            debutDeListe = append (debutDeListe, n, cpt);
            n = note[i];
        }
        else if(( change != 1) && (debutDeListe != NULL) && (note[i] != n))
        {
            debutDeListe = append(debutDeListe, n, cpt);
            n = note[i];
            cpt = 1;
        }
    }
    debutDeListe = append(debutDeListe, n, cpt);

    return debutDeListe;

}
```

Pour MIPS, c'est aussi dûr. Au début, nous voudrions ouvrir le fichier pour lire dans le Mips, mais nous n'avons pas étudié cela dans le cours, donc nous faisons des recherches sur internet, puis nous trouvons le numéro d'appel correspond à ce besoin: 13 pour ouvrire le fichier, 14 pour lire le fichier et 16 pour fermer le fichier.

open file	13	$a0 = address of null-terminated string containing filename $a1 = flags $a2 = mode	$v0 contains file descriptor (negative if error). *See note below table*
read from file	14	$a0 = file descriptor $a1 = address of input buffer $a2 = maximum number of characters to read	$v0 contains number of characters read (0 if end-of-file, negative if error). *See note below table*
write to file	15	$a0 = file descriptor $a1 = address of output buffer $a2 = number of characters to write	$v0 contains number of characters written (negative if error). *See note below table*
close file	16	$a0 = file descriptor	

Puis pour faire la liste chaînée dans le MIPS est impossible, il n'y a pas la même code dans le MIPS que C, donc nous change la moyen d'afficher l'histogramme, nous utilisons le pointeur. Pour le pointeur, nous ne savons pas comment utiliser exactement. Donc la programme est succès en compilation, mais il y a des erreur pour exécuter.

Donc nous testons chaque fonction, nous trouvons que l'erreur d'exécution n'apparaît que dans la fonction lecture.



D'après le message erreur afficher, nous pensons que la problème sortie de la chargement de la float, mais nous avons essayé aussi par "lw", cela ne marche pas non plus.

V. Les remarques

En faisant ce projet, nous trouvons qu'il y a une grande différence entre la langage C et MIPS, même s'ils peuvent d'être converties entre eux-mêmes. Avec

MIPS, nous avons besoin des pleines de registre qui signifient les adresses de variables, et des instructions internes de la fonction. Donc ils nous ne montrent pas directement les valeurs, cela perd du temps à réaliser d'une fonction, mais cela nous permet de bien comprendre les processus de l'ordinateur à exécuter d'une fonction.

Mais dans le cas de boucle, la saute de registre est difficile à tenir. En plus, dans le C, nous pouvons créer le nouveau type de structure, les listes chaînées, mais ceci ne marche pas dans le MIPS.

VI. Bibliographie

http://courses.missouristate.edu/KenVollmar/MARS/Help/SyscallHelp.html
http://gallium.inria.fr/~remy/poly/compil/1/index.html
https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c/19733-les-listes-chainees tutorial de la liste chaînée
https://emmanuel-delahaye.developpez.com/tutoriels/c/listes-chainees-c/les fonctions de la liste chaînée