INFO0947: Multiplicité du maximum

Groupe S190632: Luca Matagne,

Table des matières

1 Formalisation du problème

2 Spécification formelle

Il est importatn de déterminer la précondition et la postcondition de notre problèmes, qui seront le point de départ (pré) et d'arrivée (post) de notre raisonnement constructif.

Voici ces conditions :

— Précondition

$$N > 0 \wedge T[N] != NULL$$

— Postcondition

$$max = max(T[N]) \wedge multiplicite() = \text{nombre d'occurence du maximum}$$

3 Découpe en sous problèmes

Etant donné que le problème s'articule autour d'une seule boucle principale et que le fait de diviser le problème en deux SP qui seraient :

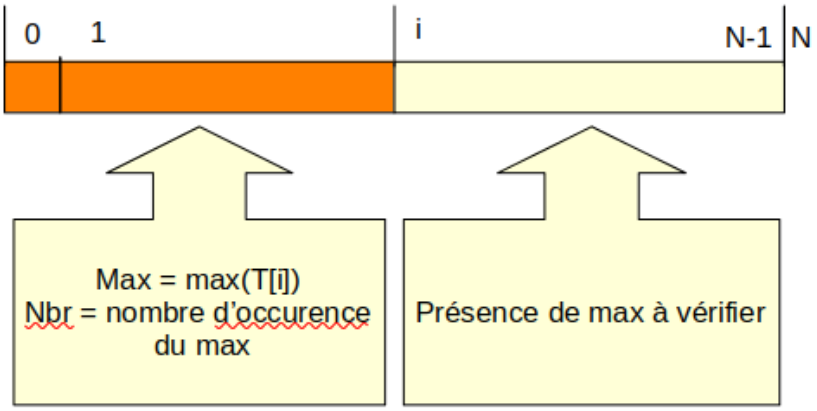
1) Recherche du maximum

2) Comptage du nombre d'occurence du maximum

amènerait a une mécompréhension (une telle découpe laisserait croire qu'il nous faut deux boucles). Je décide de ne pas diviser le problème principal en différents SP.

4 Invariant graphique et formel de la boucle

Voici l'invariant graphique de ma boucle 'while' :



Et voici l'invariant formel qui en découle :

$$N>0 \wedge T[N] != NULL \wedge 1 \leq i \leq N \wedge max = max(T[i]) \wedge nbr \geq 1$$

5 Approche constructive

Dans cette section se trouve l'approche constructive qui m'a permis de construire mon fichier "multiplicite.c"

5.1 Code d'innitialisation de la boucle

```
1
2 //N>0 ∧ T[N] != NULL
3 int i = 1;
4 //N>0 ∧ T[N] != NULL ∧ 1≤i≤N
5 int nbr = 1;
6 //N>0 ∧ T[N] != NULL ∧ 1≤i≤N ∧ nbr≥1
7 int max = T[i];
8 //N>0 ∧ T[N] != NULL ∧ 1≤i≤N ∧ max = max(T[i]) ∧ nbr≥1
```

Extrait de Code 1 – Code d'innitialisation de la boucle

5.2 Corps de la boucle

```
1
2 //N>0 ∧ T[N] != NULL ∧ 1≤i≤N ∧ max = max(T[i]) ∧ nbr≥1
3 while(i<N);
4 //N>0 ∧ T[N] != NULL ∧ 1≤i≤N ∧ max = max(T[i]) ∧ nbr≥1 ∧ i<N
5 if(T[i] > max){
6 //N>0 ∧ T[N] != NULL ∧ 1≤i<N ∧ max = max(T[i]) ∧ nbr≥1 ∧ i<N ∧ T[i]>max
7 max = T[i];
8 nbr = 1;
9 }
10 //N>0 ∧ T[N] != NULL ∧ 1≤i<N ∧ max = max(T[i]) ∧ nbr≥1
11 else if(T[i]==max){
12 //N>0 ∧ T[N] != NULL ∧ 1≤i<N ∧ max = max(T[i]) ∧ nbr≥1 ∧ T[i]=max
13 nbr++;
14 }
15 //N>0 ∧ T[N] != NULL ∧ 1≤i<N ∧ max = max(T[i]) ∧ nbr≥1
16 i++;
17 //N>0 ∧ T[N] != NULL ∧ 1≤i≤N ∧ max = max(T[i]) ∧ nbr≥1
```

Extrait de Code 2 – Corps de la boucle

5.3 Code de terminaison de la boucle

```
1
2 //N>0 ∧ T[N] != NULL ∧ max = max(T[i]) ∧ nbr≥1 ∧ i=N
3 //N>0 ∧ T[N] != NULL ∧ max = max(T[N]) ∧ nbr≥1
4 return nbr;
5 //N>0 ∧ T[N] != NULL ∧ max = max(T[N]) ∧ multiplicite = nombre d'occurence du maximum
```

Extrait de Code 3 – Code de terminaison de la boucle

6 Complexité

7 Code complet

7.1 main.c

```
1 #include <stdio.h>
2 #include "multiplicite.h"
3
```

```c

int main(){

   int T[8] = {13, -1, 16, 9, -12, 2, 4, 16};
   int max;

   multiplicite(T, 8, &max);

   printf("%d - %d\n", multiplicite(T, 8, &max), max);
}//fin main
```

Extrait de Code 4 – Main.c

7.2 multiplicite.c

```c
#include "multiplicite.h"

int multiplicite(int *T, const int N,  int *max){

    int i = 1;
    int nbr = 1;
    *max = T[0]; // le minimum pour un entier

    while(i<N){

        if(T[i]> *max){

            *max = T[i];
            nbr = 1;

        }else if(T[i]== *max){

            nbr++;

        }

        i++;

    }

   return nbr;

}
```

Extrait de Code 5 – multiplicite.c

7.3 multiplicite.h

```c
#ifndef __MULTIPLICITE_T__
#define __MULTIPLICITE_T__

/**
 * multiplicite
 *
 * @pre : N > 0 $\land$ T[N]!= NULL
 *
 * @post : max = max(T[N])
 *
```

```
 * @return : multiplicite()=nombre d'occurence du maximum
 *
**/

int multiplicite(int *T, const int N,  int *max);

#endif
```

Extrait de Code 6 – multiplicite.h

7.4 makefile

```
CC=gcc
LD=gcc
CFLAGS=--std=c99 --pedantic -Wall -W -Wmissing-prototypes
LDFLAGS=
EXEC=output

all:$(EXEC)

output: main.o multiplicite.o
	$(LD) -o output main.o multiplicite.o $(LDFLAGS)

main.o: main.c
	$(CC) -c main.c -o main.o $(CFLAGS)

multiplicite.o: multiplicite.c multiplicite.h
	$(CC) -c multiplicite.c -o multiplicite.o $(CFLAGS)
```

Extrait de Code 7 – makefile