9 章前半

その壱　未ソートデータをサーチする

文字配列

目次

```
void LinearSearch(int *A, int N, int key, int *position)
{
    int i;
    *position = -1;

    for(i = 0; i < N; i++){
        if(A[i] == key){
            *position = i;
            break;
        }
    }
}
```

文字数？

探す文字

見つからない場合は -1

見つかったら繰り返し終了

その弐　並行化した場合

```
void LinearSearch(int *A, int N, int key, int *position)
{
    int i;
    *position = -1;

    #pragma omp parallel for
    for(i = 0; i < N; i++){
        if(A[i] == key){
            *position = i;

        }
    }
}
```

追加されてる
並行化させるための一行

コア数の数だけ
この for 文が並行化
して作業する

break 消えてる

さらに改良し、３つのソースコードに分解

グローバル宣言

と

リニアサーチ関数

と

パラメータを展開するヘルパ関数

グローバル宣言

構造体
（いろいろな種類の互いに
関連するデータをまとめて，
１つのかたまりにしたもの）

```
typedef struct
{
    int *A;
    int num;
    int key;
    int threadID;
} sParam;
```

この構造体の名前

```
BOOL Done = FALSE;
```

サーチ終了を通知する
ためのフラグ

リニアサーチ関数

```
void LinearPSearch(int *A, int s, int e, int key, DWORD *position)
{
    int i;

    for(int s; i < e; i++){
        if(Done) return;
        if(A[i] == key){
            *position = i;
            Done = TRUE;
            break;
        }
    }
    return;
}
```

start から　end まで繰り返す

ヘルパ関数

```
unsigned __stdcall pSearch(LPVOID pArg)
{
    sParam *inArg = (sParam *)pArg;
    int *A = inArg->A;
    int N = inArg->num;
    int key = inArg->key;
    int tNum = inArg->threadID;

    int start, end;
    DWORD pos = -1;

    start = ((float)N/NUM_THREADS) * tNum;
    end = ((float)N/NUM_THREADS) * (tNum+1);
    if(tNum == NUM_THREAD-1) end = N;

    LinearPSearch( A, start, end, key, &pos);
    delete inArg;
    ExhitThread(pos);
}
```

受け取った構造体の値を当てはめてるだけ

配列要素数の代わりを担う

メモリの解放

この３つのコードの使用方法を
示すソースコードが次のソースコード・・・？

（何を言ってるのかよくわからない・・・）

```
for(i = 0; i < NUM_THREADS; i++)
{
	sParam *pArg = new sParam;
	pArg->A = S;
	pArg->num = NumKeys;
	pArg->key = skey;
	pArg->threadID = i;
	tH[i] = (HANDLE) _beginthreadex(NULL, 0, pSearch, (LPVOID)pArg, 0, NULL);
}

WaitForMultipleObjects(NUM_THREADS, tH, TRUE, INFINITE);

for(i = 0; i < NUM_THREAD; i++){
	GetExitCodeThread(tH[i], (LPWORD)position);
	if(*position != -1){
		printf("key = %d found at index %d\n", sKey, *position);
		break;
	}
}
if(*position == -1) printf("key = %d NOT found.\n", sKey);
```

限界です・・・

んで実行効率どうなるの？

ヘルパ関数　→オーバーヘッドを最小に
　　コールは一度だけ

しかしリニアサーチ関数で繰り返し１回に尽き条件式を
２回評価するため、それの負担が大きい。

また何度もグローバス変数へアクセスするのは時間がかかる。
↓
１０回毎にサーチ終了を判断するような条件式にすべき？

続き

・リニアサーチ関数
上から下まで順に、一致する単語があるか調べる。
見つかったらループを開放する。

・ヘルパ関数
パラメーターを展開し、
それをリニアサーチ関数へ渡して結果を
利用可能にする。

```
unsigned __stdcall pSearch(LPVOID pArg)
{
    sParam *inArg = (sParam *)pArg;
    int *A = inArg->A;
    int N = inArg->num;
    int key = inArg->key;
    int tNum = inArg->threadID;

    int start, end;
    DWORD pos = -1;
    start = ((float)N/NUM_THREADS) * tNum;
    end = ((float)N/NUM_THREADS) * (tNum+1);
    if(tNum == NUM_THREAD-1) end = N;

    LinearPSearch( A, start, end, key, &pos);
    delete inArg;
    ExhitThread(pos);
}
```

あらゆる型のデータへのポインタ

プロセッサ数のこと

unsigned ling の型

スレッドを終了させる

ここにでてきた　NUM_THREADS　ですが

定義されている型ではなく、

「自分でどうにかしろ」

というものでした。

つまり、自分で PC のプロセッサ数を

割り出さなければいけなかった……！

というわけで

OpenMP の　omp_get_num_proces() という

便利な関数があったのでそれを使います。

ついでに、下の方にある

delete を使うとエラーでるので

free 関数で代用し、メモリを開放します。

```
for(i = 0; i < NUM_THREADS; i++)
{
	sParam *pArg = new sParam;
	pArg->A = S;
	pArg->num = NumKeys;
	pArg->key = skey;
	pArg->threadID = i;
	tH[i] = (HANDLE) _beginthreadex(NULL, 0, pSearch, (LPVOID)pArg, 0, NULL);
}
WaitForMultipleObjects(NUM_THREADS, tH, TRUE, INFINITE);

for(i = 0; i < NUM_THREAD; i++){
	GetExitCodeThread(tH[i], (LPWORD)position);
	if(*position != -1){
		printf("key = %d found at index %d\n", sKey, *position);
		break;
	}
}
if(*position == -1) printf("key = %d NOT found.\n", sKey);
```

スレッドを作る関数。
3 番目にアドレスを
4 番目に渡す引数を

アラート可能な待機状態にする

DWORD 型へのポインタ

限界です