C言語の基礎9

構造体の利用

構造体と一緒に利用される関数

- メモリを自前で確保・開放する
    - `malloc( )` と `free( )` について確認してください。

> https://kaworu.jpn.org/c/%E3%83%A1%E3%83%A2%E3%83%AA%E3%81%AE%E5%8B%95%E7%9A%84%E3%81%AA%E7%A2%BA%E4%BF%9D%E3%81%A8%E8%A7%A3%E6%94%BE
>
> https://www.sejuku.net/blog/25002
>
> https://www.ei.fukui-nct.ac.jp/2022/05/31/malloc-free-2022/

- なぜ、動的にメモリを確保する必要があるのでしょうか？
- `free()` を実行しない場合、どのようなことが起きると予想されますか？

リスト構造を構造体で実装する

- リスト構造とはなにか？
    - 単方向リスト（一方向リスト）
        > https://daeudaeu.com/list-structure/
    - 単方向リストを使用した場合の、メリットとデメリットを調べてください。

- データの個数が不明な場合、はじめから配列を確保してしまうと、使わない分が無駄になってしまう。
    - そこで、必要に応じてメモリ確保を行いながら、データを格納していく
    - データの格納には、単方向リストを使用することにする。

1つの数値を格納できるような構造体

```
// 単方向リストの要素を表す構造体
typedef struct Node{
    int data;
    struct Node *next;  // 次のNodeを示すポインタ
} Node;
```

- Node型を定義し、それを使用する。
    - 本当にこれで良いか？

構造体のアドレス等を表示する

```
void display(Node *head) {
    Node *current = head;
    printf("現在のアドレス\t次のアドレス\t値\n");
    while (current != NULL) {
        printf("%p\t%p\t%d\n", current, current->next, current->data);
        current = current->next;
    }
    printf("\n");
}
```

Nodeのメモリを確保し、Nodeを作成する

```
Node *createNode(int data) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

サンプルのデータを作成し、Nodeをつなぐ

- 【list0.c】

```c
int main(int argc, char const *argv[]) {
    Node *head = NULL;

    head = createNode(100);
    display(head);

  // 何個か追加してください



    return 0;
}
```

- 正しくポインタが次のアドレスを指しているのを確認してください。

【実行結果】

```
$ ./a.out
現在のアドレス　次のアドレス　　値
0x55c8c85c22a0  (nil)   100

現在のアドレス　次のアドレス　　値
0x55c8c85c22a0  0x55c8c85c26d0  100
0x55c8c85c26d0  (nil)   200

現在のアドレス　次のアドレス　　値
0x55c8c85c22a0  0x55c8c85c26d0  100
0x55c8c85c26d0  0x55c8c85c26f0  200
0x55c8c85c26f0  (nil)   300
```

Nodeを追加する関数

- Nodeを追加する関数を作成し、動作確認をする【list1.c】
- 先頭から順にたどって行って、最後のNode（nextがNULL）を探す。
    - そのNodeのnextに追加したいNodeのアドレスを設定する

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

// 単方向リストの要素を表す構造体
typedef struct Node {
    int data;
    struct Node *next;
} Node;
```

```c
// プロトタイプ宣言
Node *createNode(int);
void append(Node *, Node *);
void display(Node *);

int main(int argc, char const *argv[]) {
    Node *head = NULL;
    Node *data;

    head = createNode(100);
    display(head);

    data = createNode(200);
    append(head, data);

    data = createNode(300);
    append(head, data);

    display(head);

    return 0;
}

Node *createNode(int data) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void append(Node *head, Node *newNode) {
    Node *current = head;

    // 終わりのNodeを探す



    // 新しいNodeを追加する



}

void display(Node *head) {
    Node *current = head;
    printf("現在のアドレス\t次のアドレス\t値\n");
    while (current != NULL) {
        printf("%p\t%p\t%d\n", current, current->next, current->data);
        // printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
```

```
}
```

先頭に追加するにはどうすれば良いか？

- `append` はリストの最後に追加したが、先頭にNodeを追加したい。
  - 関数 `prepend` を考えてください。【list2.c】
  - 動作を確認するmainも用意してください。

途中に追加するにはどうすれば良いか？

- リストの途中にNodeを追加したい。
  - 関数 `insert` を考えてください。【list3.c】
  - ただし、入れる場所はdataで指定する。
    - dataが見つかった場合はそのNodeの次に追加することにする
    - 最後まで探して、指定されたdataが無かった場合は、最後に追加する。

```
int main(int argc, char const *argv[]) {
    Node *head = NULL;
    Node *data;

    head = data = createNode(100);
    display(head);

    printf("末尾に追加\n");
    data = createNode(200);
    append(head, data);

    data = createNode(300);
    append(head, data);

    display(head);

    printf("先頭に追加\n");
    data = createNode(10);
    prepend(&head, data);

    data = createNode(20);
    prepend(&head, data);

    display(head);
```

```
    printf("途中に追加\n");
    data = createNode(-10);
    insert(head, data, 200);

    data = createNode(-20);
    insert(head, data, 999);

    display(head);

    return 0;
}
```

Nodeを削除する関数

- リスト中の特定のNode（dataで検索）を削除したい
    - 関数 `delete` を考えてください。【list4.c】
    - ただし、削除するNodeはdataで指定する。
        - 最初に見つかったNodeを削除することにする。
        - 先頭のNodeは削除できないものとする

```
int main(int argc, char const *argv[]) {
    Node *head = NULL;
    Node *data;

    head = data = createNode(100);
    display(head);

    printf("末尾に追加\n");
    data = createNode(200);
    append(head, data);

    data = createNode(300);
    append(head, data);

    display(head);

    printf("先頭に追加\n");
    data = createNode(10);
    prepend(&head, data);

    data = createNode(20);
    prepend(&head, data);

    display(head);

    printf("途中に追加\n");
```

```
    data = createNode(-10);
    insert(head, data, 200);

    data = createNode(-20);
    insert(head, data, 999);

    display(head);

    printf("途中を削除\n");
    delete (head, 100);
    display(head);

    printf("先頭を削除\n");
    delete (head, 20);
    display(head);

    return 0;
}
```

- リストから特定のNodeを取り外すだけでは、問題が生じる危険性がある。
- 安全に、長期稼働できるような修正を加えて欲しい。【list5.c】

課題

- list_prog1.c として作成
- 以下のようなデータを扱うための構造体を定義してください。
    - クラス番号（自然数で最大40）
    - 氏名（最大20バイト）
    - 国語の点数（100点満点）
    - 数学の点数（100点満点）
    - 英語の点数（100点満点）
- リスト構造で、上記データを管理するものとする。
- 以下の5人分のデータを用意しリスト化する。

番号	氏名	国語	数学	英語
1	Endou	80	80	70
20	Okita	60	95	50
4	Amano	70	70	70
9	Inoue	50	90	90
11	Ueda	85	90	60

- リストをたどって、以下の操作ができる関数を作成してください。
  - 各人の合計点を計算する `total` 関数
    - 番号、氏名、各得点、合計点を一覧で表示する
  - 各教科の最高点を探し、該当の番号と氏名を表示する `findMax` 関数
  - その他、補助的に必要な関数は適当に定義して良い。
  - 作成した関数を利用するようなmain関数を作成してください。

参考

【main】

```
int main(int argc, char const *argv[]) {
    Node *head, *data;

    head = createNode(1, "Endou", 80, 80, 70);
    data = createNode(20, "Okita", 60, 95, 50);
    append(head, data);
    data = createNode(4, "Amano", 70, 70, 70);
    append(head, data);
    data = createNode(9, "Inoue", 50, 90, 90);
    append(head, data);
    data = createNode(11, "Ueda", 85, 90, 60);
    append(head, data);

    display(head);

    printf("合計点\n");
    total(head);

    printf("国語の最高点\n");
    display_one(findMax(head, "国語"));

    printf("数学の最高点\n");
    display_one(findMax(head, "数学"));
```

```
    printf("英語の最高点\n");
    display_one(findMax(head, "英語"));
    return 0;
}
```

【実行結果】

```
$ ./a.out
No      氏名    国語    数学    英語    現在のアドレス  次のアドレス
1       Endou   80      80      70      0x5623a94282a0  0x5623a94282e0
20      Okita   60      95      50      0x5623a94282e0  0x5623a9428320
4       Amano   70      70      70      0x5623a9428320  0x5623a9428360
9       Inoue   50      90      90      0x5623a9428360  0x5623a94283a0
11      Ueda    85      90      60      0x5623a94283a0  (nil)

合計点
No      氏名    国語    数学    英語    得点計  現在のアドレス  次のアドレス
1       Endou   80      80      70      230     0x5623a94282a0  0x5623a94282e0
20      Okita   60      95      50      205     0x5623a94282e0  0x5623a9428320
4       Amano   70      70      70      210     0x5623a9428320  0x5623a9428360
9       Inoue   50      90      90      230     0x5623a9428360  0x5623a94283a0
11      Ueda    85      90      60      235     0x5623a94283a0  (nil)

国語の最高点
No      氏名    国語    数学    英語    現在のアドレス  次のアドレス
11      Ueda    85      90      60      0x5623a94283a0  (nil)

数学の最高点
No      氏名    国語    数学    英語    現在のアドレス  次のアドレス
20      Okita   60      95      50      0x5623a94282e0  0x5623a9428320

英語の最高点
No      氏名    国語    数学    英語    現在のアドレス  次のアドレス
9       Inoue   50      90      90      0x5623a9428360  0x5623a94283a0
```