プログラミング発展

2023年度2Q　火曜日5~7時限(13:45~16:30)
金曜日5~7時限(13:45~16:30)

工学院　情報通信系

尾形わかは，松本隆太郎，
Chu Van Thiem, Saetia Supat
TA:東海林郷志，千脇彰悟

第7回「データ構造」

1. 前回の復習（課題の解説）
2. データ構造とは
3. 線形リスト

※　構造体のメンバの参照方法

データ構造

データ構造

- 配列：メモリを塊で用意，ランダムアクセス可能
- 連結リスト：細切れのメモリを繋げて使う．シーケンシャル処理
- スタック：最後に入力したものを最初に処理する（LIFO）．
- キュー：先に入力したものを先に処理する（FIFO）．
- 探索木：探索を効率的に実行するために，木構造にデータを配置
- 連想配列：任意の文字列での検索が可能な配列

データの格納の方法によって，効率が変わってくる．

データ構造（配列）

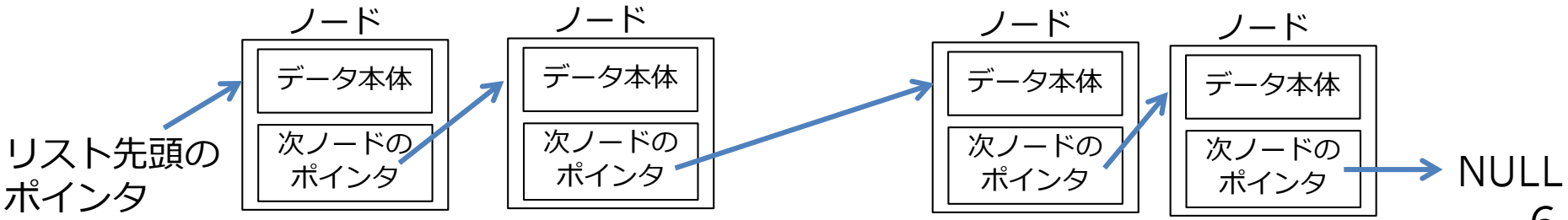
- 複数のデータに番号（0,1,2,…）が振ってある.
- 一続きのメモリ領域を確保.
- ｉ番目のデータdata[i] の読み書きは，配列サイズに関係なく，瞬時に可能. ☺（O(1)の計算時間）
- データを途中に追加する場合，
  追加するデータ以降のデータの場所を入れ替え.
  削除の場合も同様. ☹ （追加・削除時間が O(n)）
- 配列サイズ以上のデータは追加できない.
  サイズを大きめに取っておけばよいが，メモリの無駄が生じたり，メモリが確保できなかったりする. ☹

データ構造（連結リスト）

- 細切れのメモリ領域を無駄なく使える！
- 自由にサイズを増減できる！
  - 必要に応じて，malloc，freeを使って使用メモリを増減させればよい.

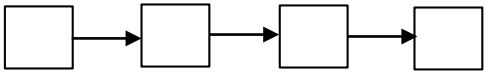
- 各データ（“ノード”と呼ばれる）は，次のメンバを持つ構造体
  - データそのもの　：　複数のメンバでもＯＫ
  - 他のノードへのポインタ

・片方向線形リストの場合

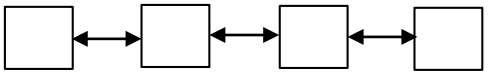


連結リストいろいろ

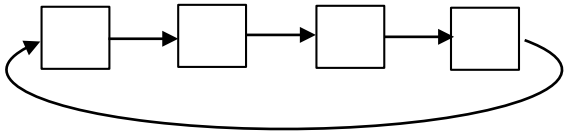
- 単方向（片方向）線形リスト：
  各ノードは，次ノードのポインタを持つ.



- 双方向線形リスト：
  各ノードは，次のノードと前のノードのポインタを持つ



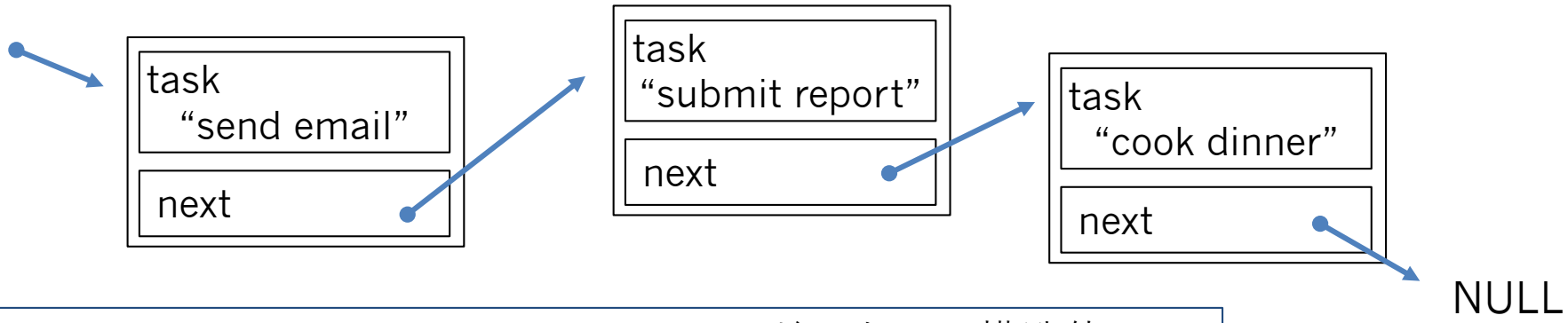
- 循環リスト：
  最初と最後がつながって輪になっているリスト.



片方向線形リストの使い方

単純な仕事の管理の例：

- やるべき仕事（文字列）をどんどんリストに追加．（いくつになるか分からない）
- 必要に応じて，現在やるべき仕事のリストを表示
- 仕事が終了したら，リストから削除．（途中にある仕事が削除されることもある）




```
typedef struct node_st{     //  ノードのための構造体
    char task[50];          // ノードに格納するデータ
    struct node_st *next;   // 次のノードへのポインタ
} node_t;
```

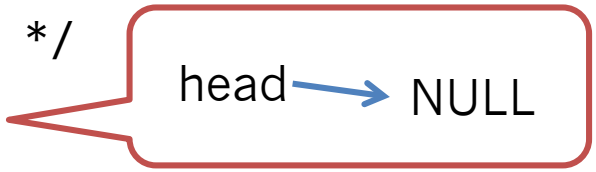
この時点ではまだ typedefが完了していないので，「node_t」は使えません

片方向線形リストの使い方

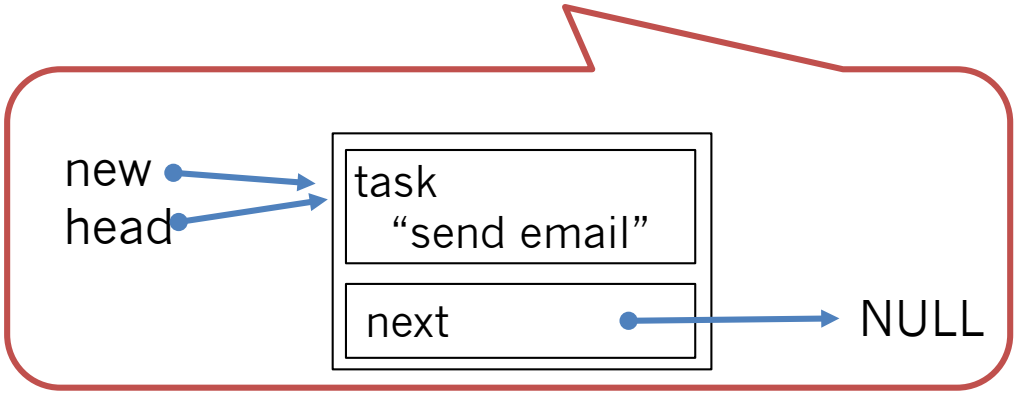
単純な仕事の管理

```
int main(void){

    /* リストにノードが一つもない状態から始める */
    node_t *head = NULL;
```



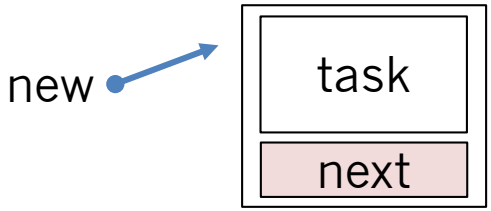

```
    /* 最初のノードを追加 （head==NULLの場合） */
    node_t *new = malloc(sizeof(node_t));
    strcpy((*new).task, "send email");
    (*new).next = NULL;
    head =new;
```



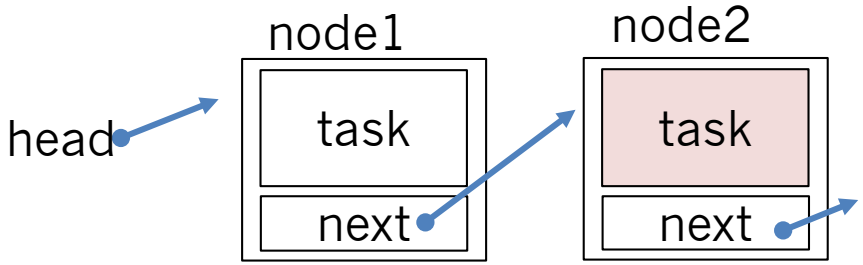
構造体のメンバの参照方法

```
(*new).next = NULL;
```
は
```
new->next = NULL;
```
とも書く。

構造体へのポインタnew が
示しているアドレスにある
構造体のメンバnext



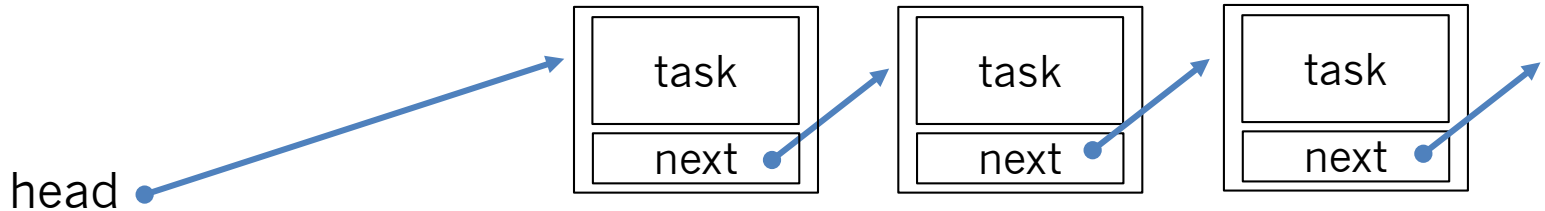
右の状況のとき、
```
node2.task
*(node1.next).task
(node1.next)->task
(*(*head.next)).task
head->next->task
```
は全て同じ値を参照する。

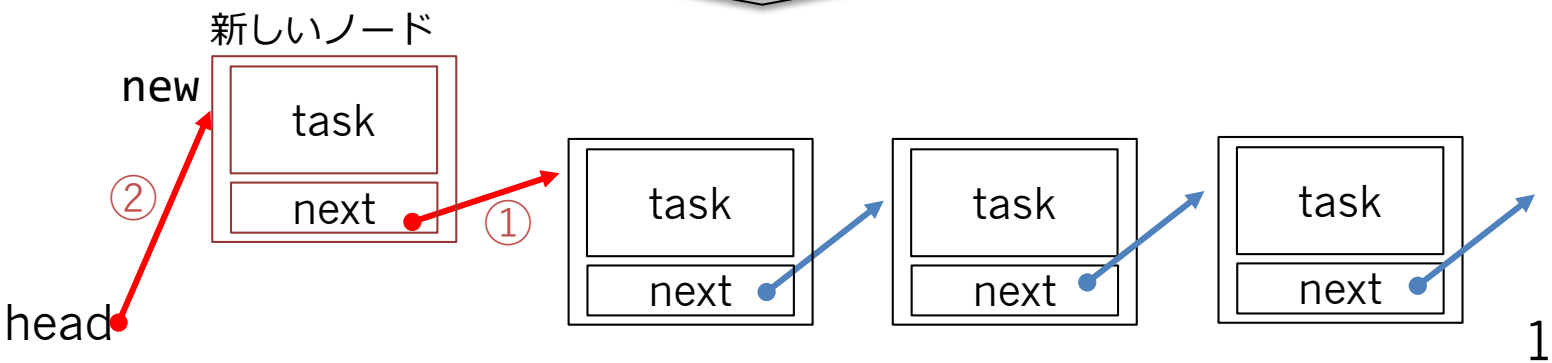


片方向線形リストの使い方（ノードの追加１）

新しいノードをリストの最初に追加する場合

```
new = malloc(sizeof(node_t));
strcpy(new->task, "submit report");
new->next = head; ①
head = new;  ②
```



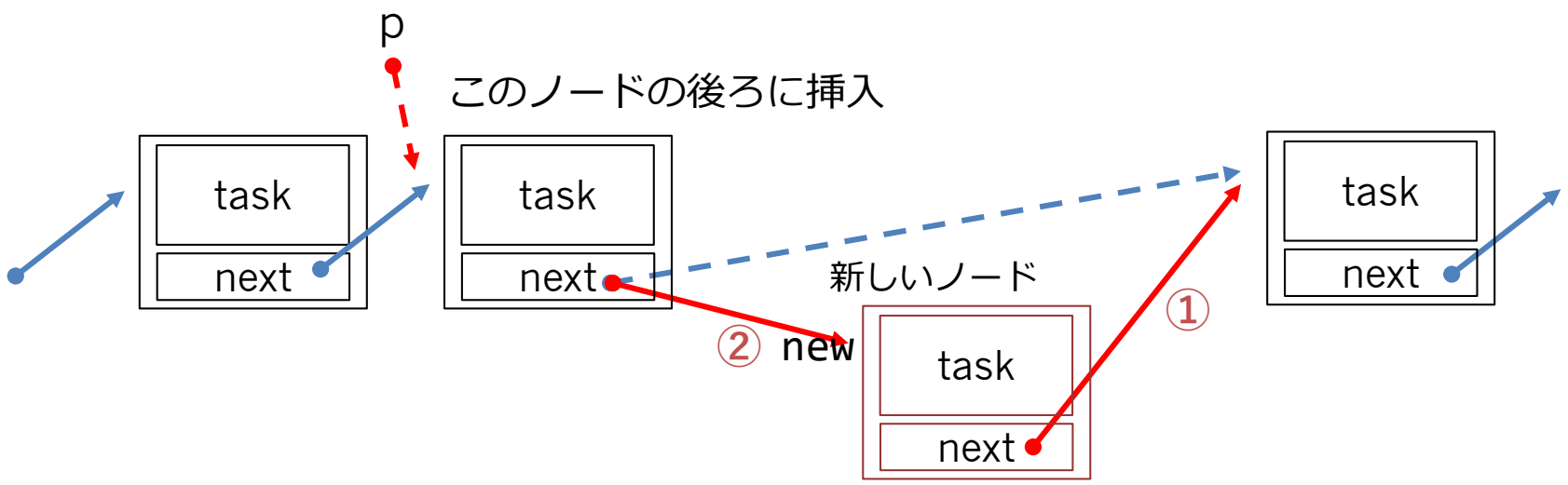


片方向線形リストの使い方（ノードの追加２）

新しいノードを、指定したノードの後ろに追加する場合

指定したノードを示すポインタを p とする

```
new = malloc(sizeof(node_t));
strcpy(new->task, “buy a notebook”);
new->next = p->next; ①
p->next = new;  ②
```



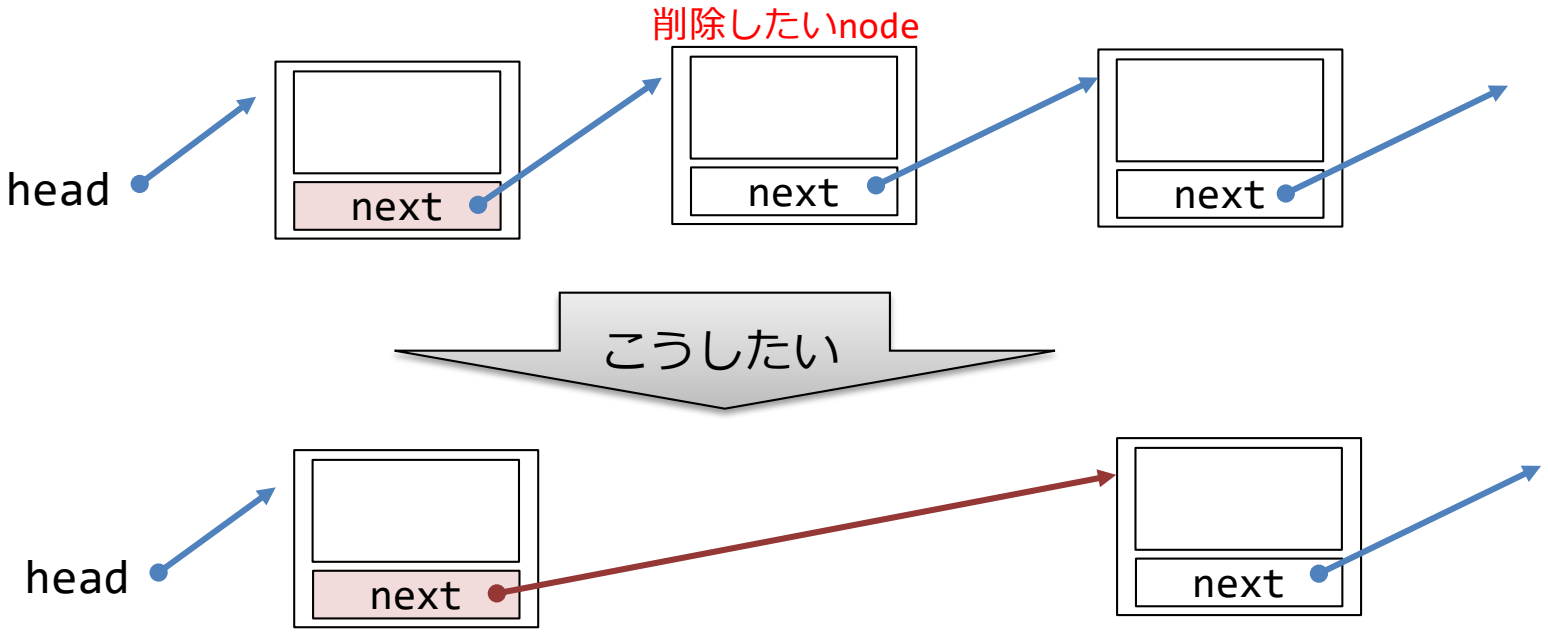
常に最後のノードのポインタを保持しておけば、「最後にノードを追加する」ことも容易。

片方向線形リストの使い方（ノードの削除）

削除するノードnodeへのアドレスが格納されているポインタのアドレス
zipnode_t **pointer_to_prev_node_next
を指定すれば，以下でできる．

`next` のアドレス．
先頭のnodeを削除する場合は &head

```
zipnode_t *node = *pointer_to_prev_node_next; // 削除するノードの場所
*pointer_to_prev_node_next = node->next; // 赤矢印を設定
free(node); node=NULL;
```



片方向線形リストの使い方（リストの表示）

リストの表示（リスト内の全てのノードの内容を表示）

```
for (node = head; node !=NULL; node = node->next)
      printf("%s ¥n",node->task);
```

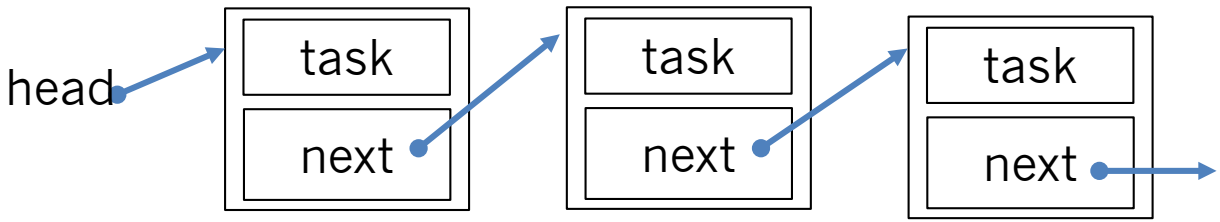
このforループは
検索にも使える

片方向線形リストの使い方（全メモリの開放）

全メモリの開放　（for 文でも書ける）

```
while (head != NULL) {
    node = head->next;
    free(head);
    head = node;
}
```

いきなり free(head)してしまうと、後ろのメモリ (head->next) にアクセスできず解放できなくなるため, nodeにメモっておく.



Ex7-1

まずは，片方向線形リストを使ってみよう.
ノードの構造体は以下のとおり定義する。（ひたすら名前を格納するリスト）

```
typedef struct node_st{
    char fullname[30];
    struct node_st *next;
} node_t;
```

struct node_st と node_t は同じ
以後は node_t を使える.

① リストの先頭アドレスhead（のアドレス）と，文字列strを引数として受け取り，新しいノードを作成し（mallocでメモリを確保する）、文字列をノードのfullnameに格納し、**リストの先頭**に新しいノードを追加する関数:

```
int add_node(node_t **head, char *str);
```

を実装しなさい. （戻り値は，追加成功なら0,失敗なら1 などとする. ）

② ３人の氏名を標準入力から受け取り、それらをadd_node関数を用いて片方向線形リストへ追加し，最後にリストの内容（３人分の氏名）を表示するプログラムを作りなさい.

提出物：
- ソースファイル（ ex7_1.c ）

実行例

```
Input > Toukou Taro
Input > Ooka Hanako
Input > Suzukake Jiro
Suzukake Jiro
Ooka Hanako
Toukou Taro
```

補足説明 1

リストの先頭を示すheadは node_t型ポインタ　（node_t *）

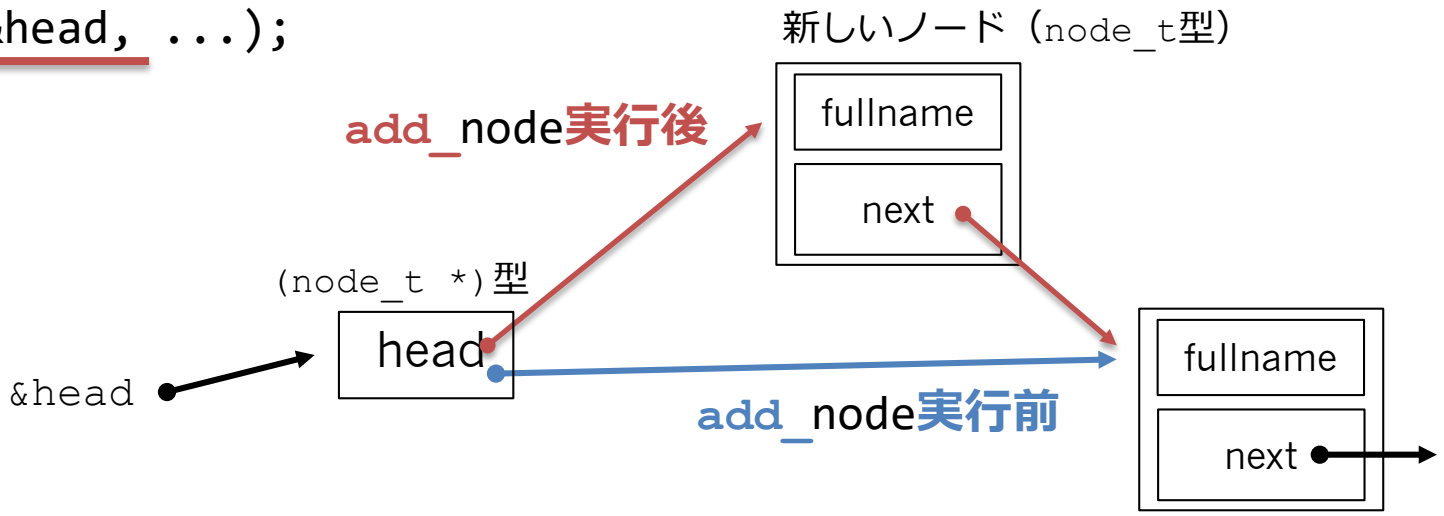
add_node関数では，引数であるheadの値が変わる．
そこで，headはアドレス渡しにする必要がある．

```
int add_node(node_t **head, char *str);
```

mainでadd_nodeを呼び出すときには

```
    add_node(&head, ...);
```

となる．



補足説明2
（スペースを含む文字列の入力）

スペースを含む文字列の入力に，scanfは使えない！

⇒　getcharを繰り返し使うか，fgetsを使う．

fgets利用例

```
  char fullname[128];
  printf("full name?");
  fgets(fullname, sizeof(fullname), stdin);
```

簡単のため128文字未満のfullnameが入力されると仮定する．

代入先のchar *型の変数

読み込む文字列の最大長

標準入力から読み込むことを示す

両方必要

Tokyo Tech（Enter）と入力すると、
fullname="Tokyo Tech¥n"となるため、最後の'¥n'を'¥0'に変更する必要がある：

```
  fullname[strlen(fullname)-1]='¥0';
```

Ex7-2

郵便番号リストのデータを，今度は線形リストに格納し、住所から郵便番号を検索してみよう。ノードの構造体は以下のとおり定義する。

```
typedef struct _zipnode {
    int zip;
    char fulladdr[FULL_ADDR_SIZE];
    struct _zipnode *next; // 次のノードのポインタ
} zipnode_t;
```

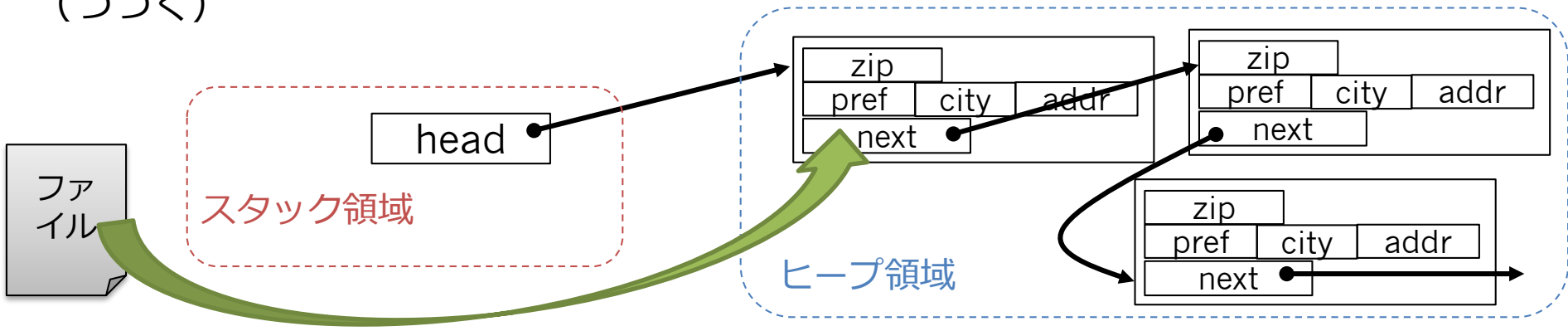
```
#define PREF_SIZE  20
#define CITY_SIZE  50
#define ADDR_SIZE  100
#define FULL_ADDR_SIZE 170
```

① リストの先頭アドレスheadと１つのデータ（郵便番号、住所）を引数として受け取り、 新しいノードを作成してデータを格納し，そのノードをリストの先頭へ追加する関数：

`int add_node(zipnode_t **head, int zip, char *fulladdr)`

を実装しなさい。戻り値は、適宜決めること。

（つづく）



Ex7-2 （つづき）

② 線形リストに格納されたデータのうち，最初のn個を出力する関数

```
int print_n_node(zipnode_t *head, int n)
```

を実装しなさい。データがn個未満の場合はあるだけ出力し，戻り値は出力したデータ数とする．

③ 住所から郵便番号を検索する関数：

```
zipnode_t *search_node(zipnode_t *head, char *fulladdr)
```

を実装しなさい。戻り値は、検索でヒットしたノードへのポインタとし，何もヒットしない場合は NULLを返す．

③ 住所から郵便番号を検索して出力するプログラムを作りなさい．
仕様は以下の通り．

- まずtokyo_all_dat.txt からデータを１行ずつ読み込み※１、add_node関数を用いて線形リストへ追加する．
  ただし，住所 fulladdr は，pref と city と addrを、スペースを挟んで連結したものとする．※２
- 線形リストへ全データを格納後、線形リストの最初の5個のノードの郵便番号（zip）と住所（fulladdr）を表示する．（格納できたことの確認）
- 次に，標準入力から住所を受け取り，search_node関数を用いて線形リスト内を検索し，ヒットした郵便番号を表示する．ヒットしなければ “no data” と表示する．

（つづく）

Ex7-2　　（つづき）

④　③のプログラムを少し変更して，住所として "exit" が入力されるまで何度でも検索できるようにしなさい.

実行結果の例

```
1002211 : TOKYO TO OGASAWARA MURA HAHAJIMA
1002101 : TOKYO TO OGASAWARA MURA CHICHIJIMA
1002100 : TOKYO TO OGASAWARA MURA ...
full address ? > TOKYO TO OGASAWARA MURA HAHAJIMA
zipcode = 1002211
full address ? > somewhere
no data
full address ? > exit
Bye!
```

Ex7-2　（つづき）

※１　tokyo_all_dat.txt からのデータの読み込みについては，以前用いた関数read_from_csvを適宜変更して使うこと.
今回はデータを配列には格納せず，その代わり線形リストへ追加すること.

※２　「pref と city と addrを、スペースを挟んで連結」は,

```
fulladddr = pref + " " + city + " " + addr
```

としたいところですが,

```
strcpy(fulladdr,pref);
strcat(fulladdr," ");
strcat(fulladdr,city);
. . .
```

のようにする必要がある.

※strcpyには、セキュリティ上の問題（範囲外アクセス）がある。これを回避する関数として strncpy 等がある。詳細はウェブや参考書等を調べてみよう。
※windowsでは、strcpyの代替としてstrcpy_sがあるが、これは採点環境で使用できないので使わないでください。

Ex7-2　　（つづき）

発展課題（余裕のある人だけやってください）

⑤　②で作った関数search_nodeを参考に，住所から郵便番号を検索して，検索でヒットしたノードのアドレスが格納されているポインタのアドレス（削除の説明スライド参照）を返す関数：

```
zipnode_t **search_pointer_to_node(zipnode_t **head_p, char *fulladdr)
```

を実装しなさい。最初のノードがヒットした場合はhead_pが返ることになる．何もヒットしなければNULLを返す．

⑥　④で作ったプログラムとsearch_pointer_to_nodeを利用して，住所から郵便番号を検索し，ヒットしたデータを削除するプログラムを作りなさい．仕様は以下の通り．

- ④のプログラムと同様にtokyo_all_dat.txt からデータを全て読み込み線形リストへ格納し，**print_n_nodeで最初の5つのデータを出力する**.
- 標準入力から住所を受け取り，**search_pointer_to_node関数を用いて**線形リスト内を検索し，検索結果の郵便番号を表示し，**ヒットしたデータを削除し，削除後にprint_n_nodeで最初の5つのデータを出力する.**（ヒットしなければ “no data” と表示し，データの削除は行わない．）
- 住所として"exit"が入力されるまで，検索して削除を繰り返す．

Ex7-2　　（つづき）

★提出物：

・④のプログラム（ ex7_2_4.c ）
　exitが入力されるまで何度でも検索できるようにする
　関数add_node, print_n_node, search_nodeを含む

（余裕があれば）
・⑥のプログラム（ ex7_2_6.c ）
　検索結果をどんどん削除する
　関数add_node, print_n_node, search_pointer_to_nodeを含む

今日の提出物 まとめ

- Ex7-1: プログラム (ex7_1.c)
- Ex7-2:
  ④のプログラム（ ex7_2_4.c ）
  ⑥のプログラム（ ex7_2_6.c ） （発展課題）

再度　注意！

以下のようなメモリの不適切な利用は、セキュリティホールになりうるため減点対象です.

- char *c; *c = ‘A’;
  ’A’を格納するメモリは確保していないのに格納してしまっている。
- mallocで確保したメモリ領域をfreeせずにプログラムが終了している．あるいは，一度freeしたメモリを，再度freeする．（p==NULLなのであれば，free(p); free(p);　と複数回free しても問題はない）
- node_t *head　を初期化しないで使う

コピペレポートについて

プログラムや考察などが他の提出者と重複している場合、不正とみなして減点および問い合わせをすることがあります