5 章_データ構造

5-1 case 1 : 単語の使用頻度を数える

この章では、より一般的なデータ構造を構築する上でのポインタの使い方を解説する。 実際にコードをたくさん書く。

5-1-1 例題の仕様について

「単語の出現頻度を数える」プログラム `word_count` を考える。

```
word_count filename
```

としたら、そのファイルに含まれる英単語をアルファベット順にソートして、各単語順に出現回数をつけて標準出力に出力する。 引数を省略した場合は、標準入力からの入力を処理する。

5-1-2 設計

大規模な開発では、プログラムを機能単位で分割することが重要になる。

1. 単語取得部 : 入力ストリームから単語を 1 つずつ取得する
2. 単語管理部 : 単語を管理する
3. メインルーチン : 1 と 2 を統括する

5-1-3 配列版

「単語管理部」のデータ構造を配列にした場合の実装

5-1-4 連結リスト版

「単語管理部」のデータ構造を連結リストにした場合の実装

5-1-5 検索機能の追加

二分検索で単語を検索する。

5-1-6 その他のデータ構造

双方向連結リスト

```
typedef struct Node_tag {
    struct Node_tag *prev; // 前の要素へのポインタ
    struct Node_tag *next; // 次の要素へのポインタ
} Node;
```

双方向連結リストは連結リストの以下の問題を解決できる。

- リストに要素を追加する時、追加する位置の前の要素がわからなければならない
- リストを要素から削除する時、削除する要素の前の要素がわからなければならない
- リストを逆順に辿ることが難しい

しかし、以下のようなデメリットもある。

- 要素ごとにポインタが 2 つ必要なのでメモリを余計に消費する
- 操作しなければならないポインタが多いので、コーディングでバグを入れやすい

木構造

```
typedef struct Node_tag {
    int n_children; // 子の数
    struct Node_tag **child; // この先に、malloc()で子へのポインタの可変長配列をつなぐ
}
```

ハッシュ

典型的なハッシュである「外部連鎖ハッシュ」では**ハッシュテーブル**という配列から、連結リストで要素を保持する。 ある要素が格納されるハッシュテーブルの添字を決めるのがハッシュ関数である。

5-2 case2 : ドローツールのデータ構造

筆者が作ったサンプルのドローツールである「X-Draw」のデータ構造だけを考えてヘッダファイルのみ作成する。

- X-Draw