

情報基礎BII

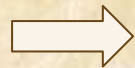
担当 柳本・野津

TA 中島

ファイル読み込みの必要性

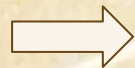
プログラムを使う段階でファイルにアクセスすることが必要

大量のデータを入力としてプログラムを実行したい



キーボードからの入力では困難
データを変えるごとにコンパイルするのは面倒

大量の実験結果を後で解析したい



画面表示ではデータの保存が困難



ファイルの入出力を利用する

ファイルの入出力

読み込み先を指定して文字列を読み込むには？

fgets()関数を利用する

fgets(配列のポインタ, 文字数, ファイルポインタ)

- ・キーボードから読み込む場合

fgets(buf, MAX_BUF, stdin);

- ・ファイルから読み込む場合

fgets(buf, MAX_BUF, fp);

ファイルポインタの指定は？

fopen()関数を利用する

fp = fopen(**ファイル名**, **モード**)

モード: 読み込み...**r**

書き込み...**w**

ファイルのオープンに失敗(ファイルがない場合など)には**NULL**を返す

ファイルのクローズ

オープンしたファイルはクローズしなくてはならない

(理由)

1. 一度にオープンできるファイルに制限
ファイルを管理するテーブルの数に制限
2. 同時にオープンするとファイルが壊れる可能性がある
同時に書き込むとファイルの整合性がとれない

ファイルの操作が終わったら必ずクローズをする

fclose関数の使い方

fclose(ファイルポインタ)

サンプルプログラム

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

```
#define MAX_BUF 256
```

```
int main(void){
```

```
FILE *fp;
```

```
char buf[MAX_BUF];
```

表示したいファイル名を書く

```
if((fp = fopen("ファイル名", "r")) == NULL){  
    printf("Cannot open file.¥n");  
    return(-1);  
}
```

← ファイルを扱うときにはエラー
処理を必ず追加すること

```
while(fgets(buf, MAX_BUF, fp) != NULL){  
    printf("%s", buf);  
}
```

```
fclose(fp);
```

```
return(0);
```

```
}
```

引数の変換

読み込まれたデータは文字列として扱われる



引数を数字として扱いたいときにはどうすればよいか？

sscanf()関数を利用

sscanf(配列のアドレス, “フォーマット”, 変数のアドレス);
文字列をフォーマットにしたがって変換し、変数に保存する

atoi()関数、atof()関数を利用

atoi(配列のアドレス);
atof(配列のアドレス);
文字列をint型(atoi)、double型(atof)に変換して、戻り値として返す

サンプルプログラム1

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

```
int main(void){
```

```
    char str1[] = "1";
```

```
    char str2[] = "2";
```

```
    int a, b, ans;
```

```
    sscanf(str1, "%d", &a);
```

```
    sscanf(str2, "%d", &b);
```

文字列を数字に変換

```
    ans = a + b;
```

```
    printf("%d\n", ans);
```

```
    return(0);
```

```
}
```

サンプルプログラム2

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

```
int main(void){  
    char str1[] = "1";  
    char str2[] = "2";  
    int a, b, ans;
```

```
    a = atoi(str1);  
    b = atoi(str2);
```

文字列を数字に変換

```
    ans = a + b;  
    printf("%d¥n", ans);
```

```
    return(0);  
}
```


メモリの動的確保

メモリの動的確保はどうして必要か？

必要な配列の大きさが分からない！

大きな配列を確保したい！

➡ 宣言による配列の確保では、(1) **あらかじめ用意されたメモリ領域**に(2) **指定された大きさのメモリ**を確保する。

(テスト)

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

```
#define MAX 10000000
```

```
int main(void){  
    int i;  
    int a[MAX];
```

```
    for(i = 0; i < MAX; i++){  
        a[i] = 1;  
    }  
  
    return(0);  
}
```

メモリの動的確保

メモリを動的に確保するには？

malloc()関数を利用する

型 *a;

a = (型 *)malloc(必要な配列の要素数*配列の要素の大きさ);

(例)

要素数10000000のint型の配列を作りたい

int *a;

a = (int *)malloc(10000000*sizeof(int));

配列の要素はint型なので、int型の変数の
大きさはsizeof(int)で求めることができる

メモリの解放

malloc関数で確保したメモリは**使い終わったら開放する**

⇒ **メモリの浪費(メモリリーク)でプログラムが不安定・強制終了**

メモリの解放には**free関数**を利用

free関数の使い方

free(malloc関数で割り当てたポインタ)

サンプルプログラム

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

```
#define MAX_BUF 256
```

```
#define MAX 5
```

```
int main(void){  
    char buf[MAX_BUF];  
    int *input, i, j, tmp;
```

必要な配列の数を引数で
指定しているのでmalloc()で
確保する

```
if((input = (int *)malloc(MAX*sizeof(int))) == NULL){  
    printf("No memory!!\n");  
    return(-1);  
}
```

使い方(実行ファイル名を
sortとして、5つの
データをソートす
る)

```
for(i = 0; i < MAX; i++){  
    printf("Input: ");  
    fgets(buf, MAX_BUF, stdin);  
    sscanf(buf, "%d", &input[i]);  
}
```

```
for(i = 0; i < MAX; i++){  
    printf("%d ", input[i]);  
}  
printf("\n");  
for(i = 0; i < MAX; i++){  
    for(j = i; j < MAX; j++){  
        if(input[i] > input[j]){  
            tmp = input[i];  
            input[i] = input[j];  
            input[j] = tmp;  
        }  
    }  
}  
for(i = 0; i < MAX; i++){  
    printf("%d ", input[i]);  
}  
printf("\n");  
free(input);  
return(0);  
}
```