外部変数

関数の外で定義される変数を外部変数（大域変数）と呼ぶ。
外部変数のスコープは広域的（プログラム全体）。全ての関数で参照可能。

```
int a=10;
double x=3.14159;

main()
{
  printf(“a = %d\n”, a);
  sample();
  printf(“%f\n”, x);
}

void sample(void)
{
  printf(“%f\n”, x);
  x += 1.0;
}
```

外部変数 a, x の宣言。変数 a, x はどの関数（main 文を含む）からも利用可能。

外部変数はどの関数からも参照可能なので、関数を独立したブラックボックスとして定義することが困難になる。

変数の隠ぺいができないため、外部変数の乱用は避けるべき。

外部変数と局所変数

```
int a=10, i;
double x=3.14159;

main()
{
  printf(“a = %d\n”, a);
  sample2();
  printf(“%f\n”, x);
}

void sample2(void)
{
  for(i=0; i<10; i++)
    printf(“%d\n”, i);
}
```

外部変数 a, x の宣言。変数 a, x はどの関数（main 文を含む）からも利用可能。

外部変数はどの関数からも参照可能なので、関数を独立したブラックボックスとして定義することが困難になる。

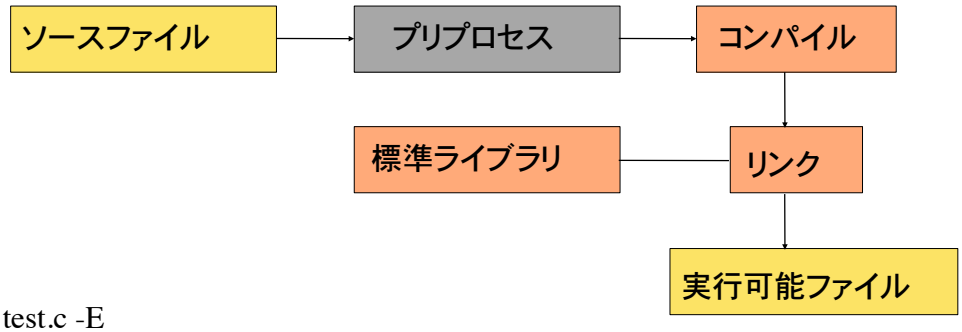
変数の隠ぺいができないため、外部変数の乱用は避けるべき。

繰り返し変数 i は sample2 の内部でのみ使用するので、局所変数として宣言すべき

プリプロセッサ

コンパイルに先立ち、ソースプログラムを指示された内容に従って書き直す処理を前処理（プリ・プロセス）という。前処理を行うプログラムをプリ・プロセッサと呼ぶ。

通常、コンパイラが呼び出されると自動的にプリプロセッサが働いてソースプログラムを修正し、修正済みのプログラムがコンパイラに渡される。従ってプリプロセッサの存在をユーザが意識することはない。



% cc test.c -E

コンパイラオプション -E により、プリプロセス後のファイルを標準出力（コンソール）に出力。コンパイル・リンクは行わない。

前処理指令

`#include`　　ファイルの取り込みを指定

`#include < ファイル名 >`　　処理系の標準ヘッダファイルを読み込む

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

`#include "ファイル名"`　　ユーザ定義のヘッダファイルを読み込む。
ヘッダファイルはソースファイルと同じディレクトリに有るものとする。

長いソースプログラムを複数のファイルに分割してコンパイルするときに用いる。共通する変数や関数の宣言をヘッダファイルにまとめて書く。

前処理指令 2

#define　　記号定数（マクロ）の定義を指定

```
#define  記号定数  文字列
```

プリプロセッサは、ソースファイルの記号定数を文字列で置き換える

```
#define SIZE 10

main()
{
  int i, score[SIZE];

  for(i=0; i<SIZE; i++)
    scanf("%d", &score[i]);

  ....
}
```

配列要素数をマクロ定義しておけば要素数の変更にも容易に対応可能

マクロ定義しておかないと、要素数に関係するソースの全ての部分を書き換えないといけない。大変。

ファイル処理

これまでは、データの入出力は標準入出力（キーボード及びコンソール）を用いてきた（シェルのリダイレクションを含む）が、入出力はファイル経由でも可能。

外部記憶装置（ハードディスクなど）に格納されたデータの集合をファイルという。ファイルを扱うために FILE という型が stdio.h に定義されている。

テキストファイル（text file）とバイナリファイル（binary file）

内容が文字と改行文字などそのまま表示可能なファイルをテキストファイル、データがバイト列となっているものをバイナリファイルという。ここではテキストファイルのみを取り扱う。

ファイルの入出力を行うには、事前、事後の操作が必要。

1) ファイルのオープン（ファイルを開く）

2) ファイルへの入出力

3) ファイルのクローズ（ファイルを閉じる）

ファイルのオープン

ファイル型へのポインタ変数宣言

```
FILE *fp;
```

fp はファイルへのポインタ。
宣言しただけでは具体的なファイルを指していない。

ファイルのオープンには、ライブラリ関数 fopen() を用いる。

```
fp = fopen("data", "w");
```

ファイル名　　モード

"w" 書き込みモード（ファイルを新規作成）

"r" 読み込みモード（ファイルが存在しないとエラー）

上記の例では、新規に data という名前のファイルが作成され、ファイルへのポインタ fp はこのファイルを指す。以後は fp を介してファイル入出力等を行う。ファイル data が既存の場合、内容は消去されて新規ファイル が作成される。

ファイル名の指定

文字列リテラルとしてファイル名を指定し、ファイルをオープンする例

```
FILE *fp;
char file_name[]="data";

fp = fopen(file_name, "w");
```

文字型配列として宣言と同時に初期化

```
FILE *fp;
char file_name[100];

scanf("%s", file_name);
fp = fopen(file_name, "w");
```

文字型配列として宣言

scanf() で文字列として入力

文字列＝文字型配列の操作により、ファイル名をプログラム内で変更可能。

ファイルオープン時のエラー処理

ファイルオープンに失敗すると fopen() は特殊な値 NULL を返す。下例のように、オープンに失敗した場合、ライブラリ関数 exit() でプログラムを正常に終了。

```
if( (fp = fopen("data", "w")) == NULL ){
  printf("Can't open file!\n");
  exit();
}
```

書き込みモードでファイルを開く場合、ディスク容量が一杯、もしくは、読み込みモードで指定したファイルが存在しない等の理由でファイルオープンが失敗する場合がある。エラー処理を行わないと、ファイルへの入出力の時点で Bus error 等のプログラムの異常終了が起こる。

ライブラリ関数 exit() はプログラムの実行を終了する。

ファイルへの入出力

ライブラリ関数 `fprintf()`, `fscanf()` を用いた入出力

それぞれ、標準入出力に関する `printf()`, `scanf()` に相当する

```
fprintf(fp, "socre is %d\n", i);
```

ファイルポインタ fp が指すファイルへ書式付き出力

```
fscanf(fp, "%d", &data);
```

ファイルポインタ fp が指すファイルからデータを整数値として読み込み、変数 data に格納。

使い方は `printf()`, `scanf()` と同じ。

ファイルのクローズ

オープンしたファイルはプログラム終了時にクローズする必要がある

ファイルのクローズ

```
fclose(fp);
```

ライブラリ関数 `fclose()` でファイルを閉じる。
一般にプログラムが終了する直前でファイルを閉じる

具体例 1

```
main()
{
  FILE *fp;
  fp = fopen("data", "w");
  fprintf(fp, "Hello!\n");
  fclose(fp);
}
```

data という名前のファイルを新規作成し、
文字列 “Hello!\n” を書き込む。

開いたファイルはちゃんと閉じること。

```
% ./a.out
% cat data
Hello!
% more data
Hello!
```

実行結果は左の通り。data というファイルが出来ていて、内容は Hello!\n である。

cat は、テキストファイルを表示するコマンド。同じようなコマンドとして more がある。

具体例 2

```
main()
{
  FILE *fp;
  int score;
  fp = fopen("data2", "r");
  while( fscanf(fp, "%d", &score)!=EOF ){
    printf("%d\n", score);
  }
  fclose(fp);
}
```

既存のファイル data2 を読み込みモード "r" で開き、ファイルに書かれているデータを整数値として読み込んで表示。

書き込みモード "w" でファイルを開くと、ファイルが新規作成されて中身が消えてしまうので注意！

ファイルオープンの際、エラー処理をしていないので、data2 というファイルが存在しないと実行時エラーで異常終了する。

具体例 2

```
% cat > data2
90
100
80
75
80
Ctrl-D
% more data
90
100
80
75
80
% ./a.out
...
```

データファイル data2 の作成。
cat コマンドを用いて、端末から直接入力。

もしくは、emacs などのエディタを用いて直接入力

ファイル入出力続き

ファイルから 1 文字ずつ文字を読み込むライブラリ関数 `getc()`, `putc()`

標準入出力に関する `getchar()`, `putchar()` に相当

```
int getc(FILE *fp);
```

fp が指すファイルから 1 文字読み込み、文字コード(int)を返す

```
int putc(int c, FILE *fp);
```

fp が指すファイルへ文字コード c に対応する文字を書き込む。書き込み時にエラーが起こると EOF を返す

具体例 3

テキストファイル data_original の内容を別のファイル data_copy にコピーするプログラム

```
main()
{
  FILE *fp_in, *fp_out;
  int code;

  fp_in =fopen("data_original", "r");
  fp_out=fopen("data_copy", "w");

  while( (code=getc(fp_in)) != EOF )}
    putc(code, fp_out);
  }

  fclose(fp_in);
  fclose(fp_out);
}
```

← コピー元のファイルは読込みモードでオープンすること。

コピー元のファイルから 1 文字ずつ読み込んで、コピー先のファイルへ書きだす。

コマンド引数

端末からプログラム(a.out)へ引き渡す引数をコマンド引数と呼ぶ。

```
% ls -a
```
コマンド ls への引数(-a がオプション)

```
% cc sample.c -lm
```
コマンド cc への引数(ファイル名とオプション)

コマンド引数は main 関数の関数頭部で定義する

```
main(int argc, char *argv[])
```

コマンド(a.out)へ引き渡される引数の個数。コマンド名(a.out)自身も引数の 1 つとなるので、この値は常に 1 以上。

与えられた引数(文字列)へのポインタの配列(引数は一般に複数あるから)

コマンド引数 2

```
main(int argc, char *argv[])
{
  printf("コマンド引数の数は %d\n", argc);
  printf("第0引数は %s\n", argv[0]);
  printf("第1引数は %s\n", argv[1]);
  printf("第2引数は %s\n", argv[2]);
}
```

これを実行すると、、、

```
% ./a.out hello! 123
コマンド引数の数は 3
第0引数は ./a.out
第1引数は hello!
第2引数は 123
%
```

上のプログラムは、配列 argv[] の添え字 2 を参照しているため、コマンド引数の数は 3 以上でなければならない。

コマンド引数 3

コマンド引数の全てを表示させるサンプルプログラム

```
main(int argc, char *argv[])
{
  int i;

  for(i=0; i<argc; i++)
    printf("%s\n", argv[i]);
}
```

argc はコマンド引数の数

コマンド引数 argv[i] を文字列として表示。i = 0 ~ argc–1

問題 1

次のプログラムを入力し、コンパイラオプション -E により、プリプロセス後のソースファイルがどのようになっているか確認せよ。

```
#include <stdio.h>

#define SIZE 10

main()
{
  int score[SIZE], i;

  for(i=0; i<SIZE; i++)
    scanf("%d", &score[i]);
}
```

#include <stdio.h> により、ソースファイルはどのようにプリ・プロセスされたか?

#define SIZE 10 により、ソースファイルはどのようにプリ・プロセスされたか?

問題 2

指定したテキストファイルをオープンして内容を表示するプログラムを作れ。このテキストファイルに含まれる文字の数を表示せよ。テキストファイル King_Lear.txt は演習時に配付する。

```
#include <stdio.h>

main()
{
  int code, count = 0;
  FILE *fp;

  fp = fopen("King_Lear.txt", "r");

  /* fp が指すファイルから1文字ずつ読み取って表示 */
  /* 文字数を計算して表示 */


  fclose(fp);
}
```

問題 3

成績が記入してあるファイルを読み込み、平均点を表示するプログラム。

なお、データファイル名は score_data、成績は 100 点満点で整数として
記入されているとする。データファイルは演習時に配付する。

データ入力にリダイレクションは使わず、ファイルをオープンして読み込むこと。

```
% ./a.out
成績処理プログラム ver 0.1, 22 July 2003
データ読み込み中...
50 人の成績を読み込みました。
平均点計算中...
平均点は 86.4 点です。
%
```

問題 4

キーボードから文字を 1 文字ずつ読み込み、Ctrl-D で中断する。読み込んだ文字をファイルに書き出すプログラムを作れ。なお書きだすファイルの名前もキーボードから入力するとする。

```
% ./a.out
ファイル名：text_file
文字を入力せよ（Ctrl-Dで終了）
Summer vacation is close at hand!
Ctrl-D
%
% cat text_file
Summer vacation is close at hand!

%
```

書き出したファイルの内容を確認せよ。

問題 5

指定したファイル名のファイルの内容を表示し、文字数と行数を表示するプログラム。エラー処理も行うこと。

```
% ./a.out
ファイル名を入力：sample.c
#include <stdio.h>
main()
{
  printf("Hello!\n");
}
文字数は 55 文字。
行数は 5 行です。
%
% ./a.out
ファイル名を入力：sample2.c
そのようなファイルはありません。
%
```

scanf でファイル名を文字列として入力。このファイルを fopen で開く。その後は 1 文字ずつファイルから文字を読み込んでいけばよい。

指定したファイルが存在しなかった場合の処理も正しく行うこと。