情報工学演習 1
第 7 回

変数のスコープ，プリプロセッサ

今日の内容

- 変数のスコープ
  - ローカル変数
  - グローバル変数
  - 静的ローカル変数
- プリプロセッサ
  - define
  - include
  - ifdef

変数のスコープ

2024/5/30

変数のスコープ

- スコープ（scope): 範囲，活動範囲
  変数のスコープ  変数が使える有効範囲

- 変数の種類によって，スコープが違う
  - ローカル変数
  - グローバル変数
  - 静的変数（static）
  - 外部変数（extern） （分割コンパイルで説明します）

ローカル変数

- main 関数や自作した関数で宣言した変数
- 有効範囲
  - 変数を宣言した関数内のみ
  - 宣言した関数の有効範囲を超えて，ほかの関数で変数を使う事はできない

```
#include <stdio.h>

void hogehoge( int n );

int main( void ){
    int a=100;
    int b=200;
    hogehoge( 3  );
    return 0;
}
```

```
void hogehoge( int n ){

    printf( "%d¥n", n );
    printf( "%d¥n", a );


}
```

グローバル変数

- 関数の外で宣言した変数のこと
- 有効範囲
  - プログラム全体
  - どの関数からも呼び出すことができる

```
#include <stdio.h>
int a=100;
void hogehoge( int n );

int main( void ){
    int b=200;
    hogehoge( 3  );
    return 0;
}
```

関数の外

```
void hogehoge( int n ){

    printf( "%d¥n", n );
    printf( "%d¥n", a );


}
```

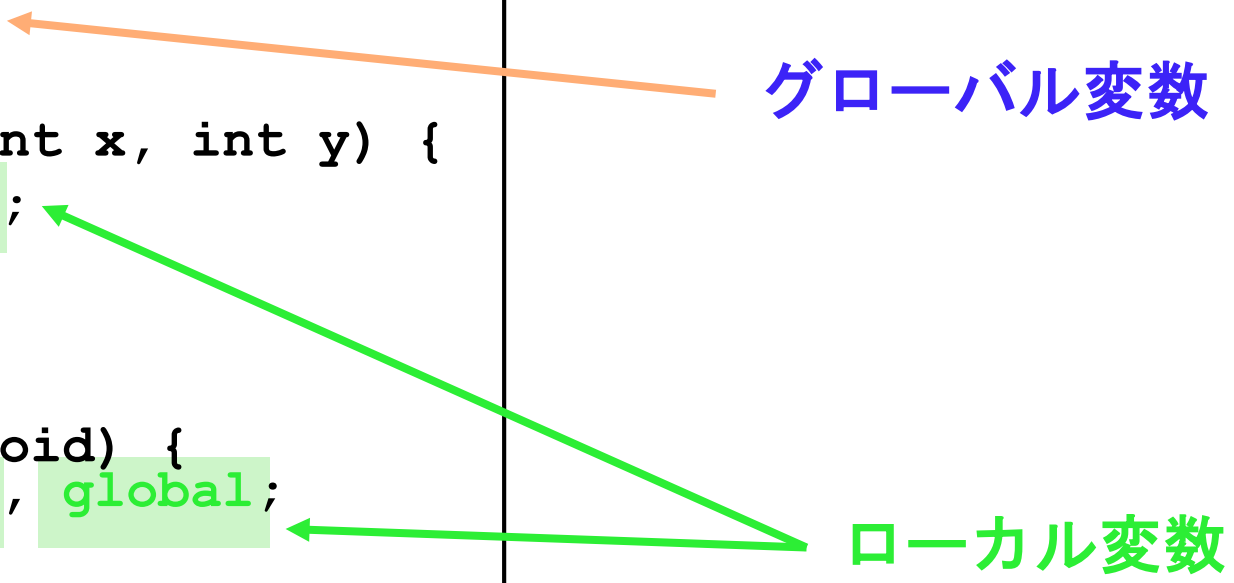
ローカル変数とグローバル変数

```
#include <stdio.h>

int global;

int func1(int x, int y) {
   int local;
   …
}

int func2(void) {
   int local, global;
   …
}
```



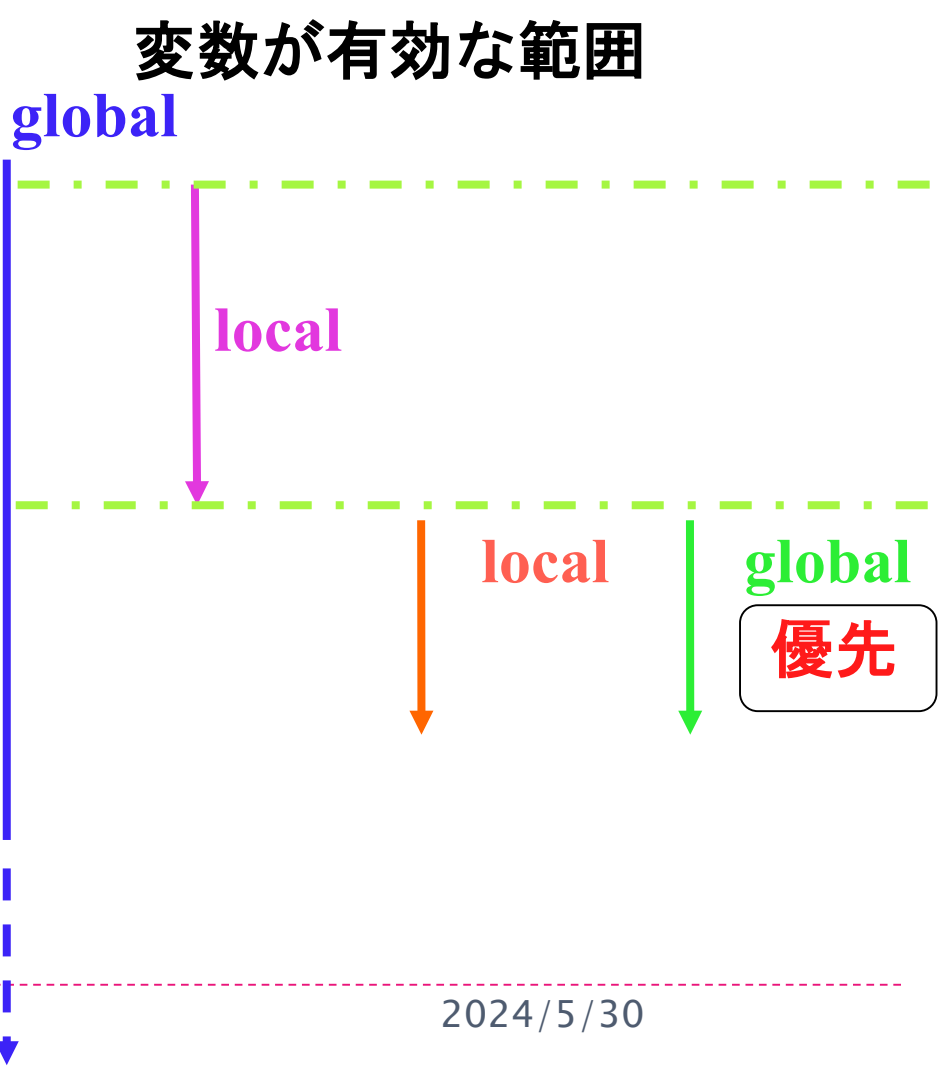
ローカル変数とグローバル変数

```
#include <stdio.h>

int global;

int func1(int x, int y) {
  int local;
  …
}

int func2(void) {
  int local, global;
  …
}
```



静的変数（static）

- 関数が何度呼ばれても変数の値を維持
- main 関数や自作した関数で宣言した変数のうち，変数の型の前に static とつけると静的変数となる
- 有効範囲
  - 変数を宣言した関数内のみ
  - 宣言した関数の有効範囲を超えて，ほかの関数で変数を使う事はできない

静的変数のサンプルプログラム

```
#include <stdio.h>

void sum( int d );

int main( void ){
    int a;
    for( a=1; a<=4; a++ ){
        sum( a );
    }
    return 0;
}

void sum( int d ){
    static int dt =0;
    dt=dt+d;
    printf( "sum=%d¥n", dt );
}
```

実行結果

```
sum=1
sum=3
sum=6
sum=10
```

最初に呼び出された時だけ
初期化

２回目以降は初期化されず，
以前呼び出された時の値を使って計算

プリプロセッサ

2024/5/30

プリプロセッサ

- コンパイラがソースコードをコンパイルする前に，一旦ソースコードに処理を施すためのプログラム
- プリプロセッサコマンド
  - `#include`
  - `#define`
  - `#undef`
  - `#if #ifdef #ifndef #else #elif #endif`
  - `#line`
  - `#pragma`
  - `#error`

書き方

- # ＋ 制御命令 ＋ パラメータリスト
  - 例：

```
#define SIZE 512
#include <math.h>
```

  - 後ろにセミコロンはつかない

#include コマンド

- 指定されたファイルをディスク上から読み込む
- ファイルの指定方法
  - <ファイル名>（例：#include <stdio.h>）
    - あらかじめ設定された標準ディレクトリからファイルを探す
  - “ファイル名”（例：#include “hoge.h”）
    - 最初にカレントディレクトリを探して，ファイルが見つからない場合は標準ディレクトリを探す

#define コマンド

- マクロの定義を行う
  - マクロ：プログラム中の文字列を，あらかじめ定義された規則に従って置換すること
- 定義の方法
  - 文字列の置き換え

    例：
```
#define TRUE 1
#define FALSE 0
#define HOGE (TRUE+FALSE)
```
    TRUE, FALSE を定数のように利用可能
  - マクロ定義
    - 関数のように引数付きで文字列を定義

    例：
```
#define mul( a, b ) ( (a)*(b) )
```
    x=mul(100, n+1);が
    x=((100)*(n+1)); として計算される

#define コマンドの長所と短所

- 長所
  - 変数の型を気にする必要がない
    - int 型でも float 型でも同じように動作する
  - 関数を呼ぶより速い
- 短所
  - 予期せぬ動作をするときがある

例：#define wa(a,b) a+b

  - wa(a,b)/2 は、関数だったら (a+b)/2 になったはずなのに、マクロだと a+b/2になる

#define wa(a,b) (a+b)とすると思った通りに動く

コンパイル時のマクロの定義

- `#define HOGE` をコンパイル時に行う場合
  gcc -D HOGE -o sample sample.c

- `#define HOGE 13` をコンパイル時に行う場合
  gcc -D HOGE=13 -o sample sample.c

#if, #else, #elif, #endif コマンド(1/2)

- マクロの定義によりコードを分岐

```
#include <stdio.h>

int main( void ){

#if MEMSIZE == 640
  char bfr[ 32000 ];
#elif MEMSIZE >= 512
  char bfr[ 30000 ];
#else
  char bfr[ 25000 ];
#endif

  printf( "The size of bfr is %lu¥n", sizeof( bfr )/sizeof( char ) );

  return 0;
}
```

配列の要素数

bfr の大きさ（byte）

char 型 の大きさ（byte）

#if, #else, #elif, #endif コマンド(2/2)

gcc -D MEMSIZE=640 -o sample sample.c

実行結果

```
The size of bfr is 32000
```

gcc -D MEMSIZE=521 -o sample sample.c

実行結果

```
The size of bfr is 30000
```

gcc -o sample sample.c

実行結果

```
The size of bfr is 25000
```

演習課題（1/4）

1. 2 つの整数をコマンドライン引数で指定させて，
   最初の整数の個数分 0 から 9 までの乱数を発生させ，
   発生させた乱数の中に 2 つめの引数と同じ値が何個
   含まれるかを数えるプログラムを作れ．

   ただし，静的変数を用いて関数が呼び出された回数を
   戻り値とする `counter` 関数を作成して，作成した
   `counter` 関数を用いて含まれる個数を数えること．

演習課題（2/4）

2. `#define` コマンドで，以下のように変数の二乗を計算するマクロを定義したが，思った通りの計算結果を返さなかった．このマクロを実際にプログラム上で実行してみて，どういう時に計算結果が間違うのか答えよ．また，どのようにマクロを定義すれば良かったか答え，プログラム上で実装せよ．
   - `#define sq(a) a*a`
   - 上記マクロと修正したマクロ (別名にする．例： `sq_2(a)`, `sq_fixed(a)` などわかりやすいもの) を 1 つのプログラムの中で実行し，結果を出力してください．
   - マクロ実行の変数の値は，2 つのマクロで計算結果が異なるものにしてください (ダメな例： `a=1` 結果が同じになる)

演習課題（3/4）

3. 以下の文字列の配列をバブルソートを使って辞書順にソートするプログラムを作成せよ.

```
char str[][256] = { “love”, “lovely”, “like”, “link”, “list”};
```

ただし，コンパイル時にDEBUG をマクロ定義した場合は，元の順番，ソートの過程，ソート結果をすべて出力し，マクロ定義をしない場合は元の順番とソート結果のみを出力するものとする．また，必要ならば文字列操作用のライブラリ関数を用いてもよい.

演習課題（4/4）

4. ユーザのキーボード入力に応じてデータの登録，データの削除，データの検索が出来る学生簿を作成せよ．学生簿は学籍番号，名字，名前，生年月日，電話番号からなり，構造体の配列でデータを持っておき，検索はどの情報でも出来るようにすること．また，プログラムを起動した際に，初期値としてコマンドライン引数で指定したテキストファイルからデータを読み込めるようにすること．
   - 例：
     - ユーザ入力１：新たに学生を登録，２：削除，３：検索
     - 上のような画面を表示して，番号を入力してもらい，番号の内容に応じて処理して，操作を続けるか終了するか聞いて，それによってプログラム終了したり再び入力待ちに戻ったりする．
   - meibo.txt を初期値読み込み用ファイルとして用意しておきます．

提出に関して（1/2）

- 提出するもの
  - ソースファイル(.c ファイル)
    - ファイル名は kadai7-学籍番号-課題番号.c
    - ソースファイルにはコメントアウトで学籍番号と氏名を記入
  - 実行結果と講義に関するコメント
    - tex で作成した PDF ファイルで，レポート中に学籍番号，氏名を含む
    - ファイル名は report7-学籍番号.pdf とする
  - 全提出ファイルを1つのフォルダに格納し，そのフォルダをzip 圧縮したファイルを提出せよ.
    - フォルダ名：report7-学籍番号
    - zip圧縮したファイル名: report7-学籍番号.zip

提出に関して（２）

- 提出期限
  - 6 月 5 日（水）23:59
- 提出方法
  - Moodle から提出
- 注意点
  - ファイル名の命名規則が間違っているものは減点する
  - Repl.itのCモードで動作しないものは採点しない(0点)