プログラミング及び演習
第3回 プリプロセッサ・型変換・演習
(2017/04/28)

講義担当
大学院情報学研究科知能システム学専攻
教授　森　健策

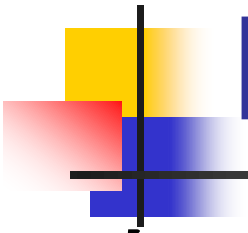
演習担当
大学院情報科学研究科メディア科学専攻
助教　小田　昌宏

本日の講義・演習の内容

- 前回の復習 (変数, 関数, 再帰)
  - 復習します。
- プリプロセッサ
- 演習
- 講義・演習ホームページ
  - http://www.newves.org/~mori/17Programming

出席

- NUCTにて出席アンケートに回答する
  - www.media.nagoya-u.ac.jp
  - 回答1には「今日の合言葉」
  - 回答2には「GWはどうする?」
  - 回答3には「先週の内容について一言」
- 出席テストは0点以上の点数であれば問題ありません。
  - 自動採点を行っていません



関数

- ある一連の手続きを一つの文で呼び出し実行する機能
  - サブルーチン
  - 副プログラム
  - 手続き (procedure)

関数の使用例

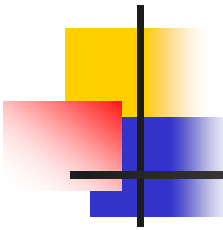
```
#include<stdio.h>

int abs(int x)
{
   if(x>=0){
      return(x);
   }else{
      return(-x);
   }
}
```

```
main()
{
    int x,y,z;
    y=-5;  z=7;
    x=abs(y);
   printf( "abs(%d)=%d",
                    y,x);
    x=abs(y+z);
   printf( "abs(%d)=%d",
                    y+z,x);
}
```

関数プロトタイプ (教 p.35-)

- 関数を定義する前に呼び出す場合
  - 関数･引数の宣言をする
  - コンパイル時に呼び出し方のチェックがなされる
- 記述法
  - <関数の型> <関数名>(<型> <引数1>,
    <型> <引数2>,…,
    <型> <引数3>);
  - 関数宣言の部分と全く同じ
  - 引数1-3は省いてもOK



Call by value (値による呼び出し) (⇔ Call by referene) (KR p.34-)

- Cではすべての関数の引数が値で呼び出される
  - 呼び出し元の変数ではなく、一時変数に値がコピーされて呼び出される
- 呼ばれた関数は呼び出した関数側の変数の値を変えることができない
- 一時的にプライベートな変数を変えることは可能

局所変数 (教p.36-)

- 関数の内部で宣言された変数　＝ 局所変数
  - その関数内部でのみ有効
  - 外部からアクセスすることは不可能
  - 異なる関数中で同じ変数名前を使用可能
  - 仮引数も局所関数
  - 関数が呼び出されるときに現れ、終了後消える
    - 関数の実行が終了すると変数の領域が開放される

大域変数 (教p.36-)

- 関数の外部で宣言された変数＝大域変数
  - すべての関数において有効
  - 大域変数と同じ変数名で局所変数を宣言したときはその関数内では局所変数が有効
  - プログラミングのときは最小限とすることが望ましい
  - グローバル変数、広域変数などとも記される

講義内課題 (lec2-var.c)

```
#include <stdio.h>
int x,y;
void fa()
{
    int x,i;
    i=0; x=1; y=2;
}
void fb()
{
    int i;
    i=0; x=1; y=2;
}
main()
{
    x=10;
    y=20;
    fa();
    printf( "%d %d¥n", x,y );
    fb();
    printf( "%d %d¥n", x,y );
}
```

(関数内での)静的変数 (教 p.46)

- 関数内で保持される変数 = 静的変数
- 宣言時にstaticをつける
  - static int k;
- 関数呼び出し終了後も変数の値が保持される

講義内課題 (lec2-stat.c)

以下のプログラムの動作を確かめよ　cの値はどうなるか?

```
#include <stdio.h>
void count();
main()
{
    int i,k;
    for(i=0;i<=10;i++){
        count();
    }
}
```

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

再帰呼び出し (教 p.39)

- 自分自身を関数内で(直接的・間接的)に呼び出すことができる
- 再帰の形で書かれたアルゴリズムを実装するのに便利
- スタックオーバーフローに注意
- フィボナッチ数列
  - fib(0)=1;
    fib(1)=1;
    fib(i)=fib(i-1)+fib(i-2);

数字を文字列として印字
講義内課題 (lec2-printd.c)

以下のプログラムの動作を確かめよ. printfはどのように呼び出されるか

```
#include <stdio.h>
void printd(int n)
{
  if(n<0){
    putchar('-');
    n=-n;
  }
  if(n/10){
    printd(n/10);
  }
  putchar(n%10+'0');
}
```

```
main()
{
  int num;
  printf("num = ");
  scanf("%d",&num);
  printd(num);
}
```

講義内課題 (lec2-fib.c)

1. num=100としたとき何回fibが呼ばれるかカウントしてみよ
2. limit stacksize 100k としてから num =100として実行してみよ

```
#include <stdio.h>
int fib(int i)
{
    printf( "enter fib(%d)¥n",i);
    if((i==0)||(i==1)){
        return(1);
    }else{
        return(fib(i-1)+fib(i-2));
    }
}
```

```
int main()
{
    int num;
    printf("num ¥n");
    scanf("%d",&num);
    printf("fib(%d)=%d¥n", num,
        fib(num));
    return 0;
}
```

プリプロセッサとは

- コンパイルのための前処理
  - Preprocessor
  - man cpp より　(Sun Solaris 9)
    - cpp は、C 言語プリプロセッサです。cpp は、cc(1B)コマンドで開始される C プログラムのコンパイルにおける最初のパスとして起動されます。ただし、cpp は、他のSunコンパイラの第1パスプリプロセッサとしても使用できます。

プリプロセッサとは

- 言語仕様に準拠したコンパイル処理が行われるのではなく，文字列処理的なことが行われる
  - #include <a.h>
    - #includeの位置にa.hファイルをそのまま挿入
  - #define STATE 1
    - このdefine以降に現れるSTATEという文字列を1で置換

ファイル読み込み - #include

- 外部関数宣言・#defineの集まりをまとめて取り扱うのに用いる
- 任意の行に挿入可能．挿入箇所をfilenameの内容で置き換える

ファイル読み込み - #include

- #include "filename"
  - #includeが記述されているソースファイルと同じディレクトリからfilenameを読み込む
  - 見つからない場合，<>で囲ったときと同じ動作をする
- #include <filename>
  - コンパイラに指定されているインクルードファイルパスからfilenameを読み込む

#include <..>

- 以下のように記述すると

gcc -o test test.c -I/home/mori/SampleInclude

#include <filename.h>を探す際，/home/mori/SampleIncludeの中も探してくれる

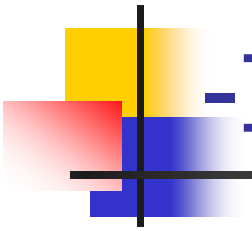
/home/mori/TestProg/test.c

```
#include <stdio.h>
#include <abc.h>
int abs(int x)
{
    printf( "abc = %d¥n", ABC)
}
```

/home/mori/SampleInclude/abc.h

```
#define ABC 100
```

-Iオプションがないとコンパイルエラー → abc.hを見つけることができない



-Iがない時(デフォルト)はどこを探す?

- gccの場合
  - /usr/include
  - ${PREFIX}/アーキテクチャ名/include

#includeはどんなときに使う?

- 大きなプログラムで同じプロトタイプ宣言・マクロ定義を利用するのに用いる
- 分割コンパイル(関数毎にファイルを分割しコンパイル)するときに極めて有効　→　後の講義にて取り扱う

マクロの置換 #define

- 文字列(token)を置換テキストで置き換える

  #define name 置換テキスト

- #defineは定義された位置から
  ソースファイルの最後まで有効
- 以前に定義されたtokenを再度定義可
- ""で囲まれた文字列は置換されない

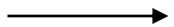
#defineの例 (lec3-macro1.c)

```
#include <stdio.h>
#define YES 1
#define NO 0
main()
{
    int ans;
    printf( "ans =" );
    scanf("%d", &ans);
    if( ans== YES)
        printf( "YES¥n" );
    else if(ans==NO)
        printf( "NO¥n" );
}
```

引数付マクロ

- 関数のようにマクロを定義可能
- 例
  - #define SUM(A,B) ((A)+(B))
  - #define MAX(A,B) ((A)>(B) ? (A) : (B))
  - #define MUL(A,B,C) ((A)*(B)*(C))

```
#define add(a,b) (a+b)
...
c= add(1,2);
```



```
c=(1+2);
```

引数付マクロの注意点

- 単なる文字列の置き換えであることに注意
- #define SQUARE(A) A*A
  は正しい?
- 括弧を使って範囲を明示!!

#defineの使用例 (lec3-macro2.c)

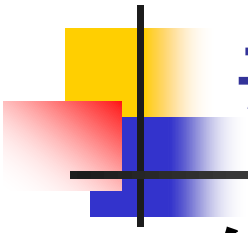
```
#include <stdio.h>
#define MUL(A,B) (A)*(B)
main()
{
   int a = 10, b=5, c;
   c = MUL(a+b,b+a);
   printf( "c = %d¥n", c );
}
```

()がないとどうなるか？　(lec3-macro3.c)

```
#include <stdio.h>
#define MUL(A,B) A*B
main()
{
   int a = 10, b=5, c;
   c = MUL(a+b,b+a);
   printf( "c = %d¥n", c );
}
```

#付パラメータ名

- パラメータ名の前に#記号がついている場合，その組み合わせは引用符付文字列に変換される
- 例:
  - #define dp(exp) printf(#exp "%d", (exp))
  - dp(a/b);として使用すると
    printf("a/b" "%d", (a/b));
    に変換される



##演算子

- マクロ展開の最中に実引数を連結
- 例:
  - #define paste(front, back) front##back
  - paste(name,1) として使用すると name1 に変換される

条件付コンパイル

- ソースコードの一部を選択的にコンパイル可能とする
- #if
- #else
- #elif
- #endif
- #ifdef
- #ifndef

条件付コンパイルの構文

```
#if 定数式
   文の並び
#endif
```

```
#if 定数式
   文の並び
#else
   文の並び
#endif
```

```
#if 定数式
   文の並び
#elif
   文の並び
#elif
   文の並び
#endif
```

使用例 (lec3-macro4.c)

```
#include <stdio.h>
#define DEBUG 1
main()
{
    int i,j=0;
    for(i=0;i<100;i++){
#if (DEBUG == 1)
        printf("i=%d¥n",i);
#elif (DEBUG >=2)
        printf("i=%d j=%d¥n",i,j);
#endif
        j += i;
    }
}
```

条件付コンパイルの構文(続き)

```
#ifdef マクロ名
    文の並び
#endif
```

```
#ifndef マクロ名
    文の並び
#endif
```

使用例 (lec3-macro5.c)

```
#include <stdio.h>
#define DEBUG
main()
{
    int i,j=0;
    for(i=0;i<100;i++){
#ifdef DEBUG
        printf("i=%d j=%d¥n",i,j);
#endif
        j += i;
    }
}
```

組み込みマクロ

- __LINE__
  - 現在コンパイルされているソース行の行番号
- __FILE__
  - 現在コンパイルされているファイルのファイル名
- __DATE__
  - 現在のシステム日付
- __TIME__
  - プログラムのコンパイルを開始した時間
- __STDC__
  - コンパイラがANSI Cに準拠していれば1

使用例 (lec3-macro6.c)

```
#include <stdio.h>

main()
{
    printf( "%d %s %s %s", __LINE__, __FILE__,
                    __DATE__, __TIME__ );
}
```

cppでの変換を見てみよう

- 以下のプログラムを入力（macrotest.c)

```
#include <stdio.h>
#define MAX(A,B) ((A)>(B) ? (A):(B))
#define ABC 2
main()
{
    int x=2, y=5;
    printf( "max = %d¥n", MAX(x,y));
    printf( "ABC =%d¥n", ABC);
}
```

- cpp macrotest.c macrotestafter.c

型変換

- 式を評価する場合暗黙の型変換が行われる
- 優先順位の高い演算子から順に行われ，最終的な結果は式中で最も高い型となる
- 通常型算術型変換
  - long double > double > float >unsigned long > long > unsigined int > int
- 汎整数拡張
  - int型よりビット長の小さい整数値(short型, char型)の整数値をint型またはunsigned int型に変換
- 代入式の場合は，上の変換規則で右辺の値を計算し，左辺の型に変換して代入

型变换

- 例
  - int n;
    char c='A';
    n=c;
  - int n,m=3;
    n=m*1.5;
  - int n=255;
    signed char c;
    c=n+2;

式における型変換の例

```
#include <stdio.h>
main()
{
    int i;
    float f;
    i = 10;
    f = 23.29;
    printf( "%f¥n",i+f);
}
```

```
#include <stdio.h>
main()
{
    float f;
    f = 100.0/(10/3);
    printf( "%f¥n",f);
}
```

それぞれどんな値が出力されるか?

代入における型変換の例

```
#include <stdio.h>
main()
{
    char ch;
    int i;
    i = 1000;
    ch = i;
    printf( "%f¥n",ch);
}
```

```
#include <stdio.h>
main()
{
    int i;
    float f;
    f = 1234.0098;
    i=f;
    printf( "%f %d",f,i);
}
```

それぞれどんな値が出力されるか?

型キャスト

- 変数の型を一時的に変える方法
- (<型>) <変数>
- 例
  - float f;
    f = 100.2;
    printf("%d", (int)f);
  - double v=1.5;
    printf("%f¥n", v);
    printf("%d¥n", v);
    printf("%d¥n",(int)v);