関数の作成と変数のスコープ（数学購読）

５月１６日課題（６月７日まで）

課題１ $f(x) = ax^3 + bx^2 + cx + d$ を計算して返す関数を作れ.

課題２ 与えられた文字列の字数を計算してそれを返す関数を作れ.

課題３ $(f(x,y), g(x,y)) = (x\cos(y) + y\sin(x), x^2 + y^2 + xy)$ を計算する関数を作れ.

課題３ ４次元ベクトルの和・差・内積・スカラ倍の計算を実行する関数を作れ.

課題４ ４×４行列の和・差・積・スカラ倍の計算を実行する関数を作れ.

実習で利用するサンプルプログラム

prog4-1.c （関数作成の基本）

```
/*
   関数の作り方について学ぶ．一般に関数には定義域と値域があるので，
   それを定義しなければならない．数学関数と違う点は定義域や値域が
   必ずしも数だけとは限らない点である．prog4-1.c では関数の定義だけ
   して，演算の中身は後で書く．
*/
#include <stdio.h>
#define NUM 200
/*
   ポイント１： 関数の定義の書式は以下の通り 関数の戻すデータ型
   (戻り値という) 関数名 ( 関数に渡すデータ型 (引数), ...)
```

```
 */
/*
  ポイント２： この例では実数二つから実数を一つ返す関数 f を定義.
*/
double f( double, double  );
/*
  以前説明したように，main も関数である．ただし，この main 関数は
  戻り型，渡し型ともにあらかじめ定義されているものである
*/
int main( int argc, char *argv[] )
{
  int i,n=NUM;
  double a;
  double x[NUM];
  a = 3.1;
  x[0] = 0.2;
  for(i=0;i<n-1;i++){
    x[i+1] = f( x[i], a );
  }
  for(i=0;i<n;i++){
    printf( "%d %lf\n", i, x[i] );
  }
  exit(0);
}
/*
  main 関数の後に関数 f の内容を定義．この関数は f(x,a)=ax(1-x)
  の値を返す.
*/
double f( double p, double q )
{
  double ret;
  ret = q*p*(1-p);
  return(ret);
}
```

prog4-2.c （関数と変数のスコープ）

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

```
#define NUM 200
/*
  ポイント１：  前の例と違いmain関数の前であれば，関数の定義と中身を
  同時に書くこともできる
*/
/*
  ポイント２：この関数内の変数名 x，a，i などはmain関数でも同じ
  名前の変数名が使われているが，C言語では変数名はその関数内
  (厳密には{}で囲まれた範囲の中) 通用するように設計されているので，
  同じ名前を使って良い．

  これを変数の「スコープ範囲」と呼び，局所 (Local) 変数とよぶ．
  これと反対に大域変数というのがあるが，これについては次の
  プログラムで学ぶ．

  逆に言うとmain関数でのi,jの値が何かについて関数fの中では
  知らないことになるので，関数の計算に必要なデータはすべて
  引数 (あるいは大域変数) として関数に渡さなければならない．
*/
double x_5( double x , double a )
{
  int i;
  double ret=1;
  for(i=0;i<5;i++){
    /* これは ret = ret*x の略記 */
    ret *= x;
  }
  ret *= a;
  return(ret);
}
int main( int argc, char *argv[] )
{
  int i,n=NUM;
  double a;
  double x[NUM],y[NUM];
  a = 3.0;
  for(i=0;i<n;i++){
    x[i] = (double)i/(double)n;
```

```
    y[i] = x_5( x[i], a );
  }
  for(i=0;i<n;i++){
    printf( "%lf %lf\n", x[i], y[i] );
  }
  exit(0);
}
/* main 関数の後に関数 f の本体を定義する */
/* この関数は f(x,a)=a*x*(1-x) の値を返す */
double f( double p, double q )
{
  double ret;
  ret = q*p*(1-p);
  return(ret);
}
```

prog4-3.c （大域変数と局所変数 call by values）

```
/*
 ここでは大域変数と関数の Call by values について学ぶ.
*/
#include <stdio.h>
int i;
/*
   ポイント１：このように大域変数は main 関数の外で定義する.
   こうすると，このプログラムを通じて 整数 i というのは
   全ての関数から参照され, データも書き換えられる.

   そのため大域変数の変数名として採用した i という文字
   はプログラムの他のどの部分でも再定義することはできない.
*/
/*
  ポイント２： 関数の戻り値として void（無）を指定した場合は
  なんの値も戻り値として返さないことを意味する.
*/
void global_local( int d )
{
  /* 大域変数に２０を代入 */
```

```
  i = 20;
  /* 引数として貰った d(main関数ではjと名付けていた)
     に１０を代入 */
  d = 10;
  return;
}
int main( int argc, char *argv[] )
{
  /* main の関数の中だけで有効な局所変数 j */
  int j=100;

  /* 関数呼出し前の大域変数 i  と局所変数 j の値の表示 */
  printf( "i = %d, j = %d\n", i, j );

  /* 関数の呼出 */
  global_local( j );

  /* 関数呼出後の大域変数 と 局所変数の表示 */
  printf( "i = %d, j = %d\n", i, j );

  exit(0);
}
```

prog4-4.c （関数の呼び出し Call by address）

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

/*
  これまでで見たように局所変数は関数の呼出によって
  変化を受けないため，関数の計算結果を関数を呼び出した
  側の局所変数に反映させるためには，戻り値として返す
  必要がある
*/

/*
  しかし、関数の戻り値としては変数型を一つしか指定でき
  ないので，ベクトルや行列などの計算結果を戻り値とする
```

```
  ためには，様々な工夫が必要である．

  C 言語ではこれに対して主に二つの方法を提供している．

  1．構造体とポインタ
  2．引数のアドレス渡しとポインタ

  ポインタは C 言語固有の概念であり，これの理解をもって
  C 言語を理解したといえる．次回からはポインタについて
  詳しく学ぶが今回は 2 について (特に理由を説明せず) 学ぶ．
*/

/* 関数 f1 はこれまでと同様に値渡し (Call by values) */
double f1( double x )
{
  double ret;
  x = 5.0;
  ret = x;
  return(ret);
}
/*
  関数 f2 はアドレス渡し (Call by address)
    "double*" は実数型へのポインタを意味する．
*/
double f2( double *x )
{
  double ret;
  *x = 5.0;
  ret = *x;
  return(ret);
}
/*
  以下の二つの関数 f3,f4 は配列を引数に渡す方法である
 */
double f3( double* a, int n )
{
  int i;
  double ret=0;
```

```
  for(i=0;i<n;i++){
    a[i] = i;
    ret += a[i];
  }
  return(ret);
}
/* もう一つの方法 */
double f4( double a[], int n )
{
  int i;
  double ret=0;
  for(i=0;i<n;i++){
    a[i] = 2*i;
    ret += a[i];
  }
  return(ret);
}
int main( int argc, char *argv[] )
{
  int i;
  double x[10],y;
  double ret;

  /* 初期化 */
  y = 10.0;
  for(i=0;i<10;i++){
    x[i] = (double)i*10;
  }
  /* 関数呼び出し前の値 */
  printf( "Before calling the functions:\n ");
  printf( "y = %lf\n", y);
  for(i=0;i<10;i++){
    printf( "x[%d] = %lf\n", i, x[i]);
  }
  printf( "\n" );
  /* 値呼び出し f1 */
  ret = f1( y );
  printf( "After calling the function f1: ");
```

```
  printf( "y = %lf, ret = %lf\n\n", y, ret);
  /* アドレス呼び出し f2 */
  /* &をつけてyの変数の格納されているメモリのアドレスを渡す */
  ret = f2( &y );
  printf( "After calling the function f2: ");
  printf( "y = %lf, ret = %lf\n\n", y, ret);
  /*  配列の呼び出し f3 */
  /*  配列 x[i] の場合は 単にx と書けばそれは配列の格納
      されているアドレスを表す */
  ret = f3( x, 10 );
  printf( "After calling the function f3: ");
  for(i=0;i<10;i++){
    printf( "x[%d] = %lf\n", i, x[i]);
  }
  /*  配列の呼び出し f4 */
  ret = f4( x, 10 );
  printf( "After calling the function f4: ");
  for(i=0;i<10;i++){
    printf( "x[%d] = %lf\n", i, x[i]);
  }
  exit(0);
}
```