アルゴリズム論 2&3

概要

- アルゴリズムの定義
- C言語の復習
- データ構造について
- 計算量について
- 再帰呼び出し

素数の抽出

- 3種類のアルゴリズムによる計算量の違いに注目
- 2～100までの素数を抽出する

(1)単純版

自分より小さい数で順番に除算を行う

自分自身でのみ割り切れた場合は素数

自分以外の数で割り切れた場合は素数ではない

(2)改良版1

奇数のみを対象にする

自分より小さい素数で除算を行う

自分自身でのみ割り切れた場合は素数

自分以外の数で割り切れた場合は素数ではない

(3)改良版2(エラトステネスのふるい)

奇数のみを対象にする

自分の平方根より小さい素数で除算を行う

自分自身でのみ割り切れた場合は素数

自分以外の数で割り切れた場合は素数ではない

単純版(prime1.c)

ソースファイル

```
#include <stdio.h>

int main(void)
{
      int   i,n;
      int         N=100;        /* 最大の数 */
      int         counter=0;   /* 除算の回数 */

      for (n=2;n<=N;n++)     {
            for (i=2;i<n;i++)    {
                  counter++;
                  if (n%i==0)    /* 割り切れたら素数でない   */
                  break;
            }
            if (n==i)   printf("%d¥n",n);   /* 最後まで割り切れない */
      }
      printf("Total number of calculation : %d¥n",counter);

      return(0);
}
```

実行結果

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
Total number of calculation : 1133
```


改良版1(prime2.c)

ソースファイル

```
#include <stdio.h>
int main(void)
{
      int   i,n;
      int   N=100; /* 最大の数 */
      int   prime[500]; /* 素数を格納する配列 */
      int   ptr=0;    /* 抽出した素数の数 */
      int   counter=0; /* 除算の回数 */

      prime[0]=2; /* 最初の素数   */
      ptr++;

      for (n=3;n<=N;n+=2)  {  /* 奇数のみを対象   */
            for (i=1;i<ptr;i++) {  /* すでに抽出した素数で除算   */
                  counter++;
                  if (n%prime[i]==0)   /* 割り切れたら素数でない   */
                        break;
            }
            if (ptr==i)
                  prime[ptr++]=n;      /* 最後まで割り切れない */
      }
      for (i=0;i<ptr;i++) printf("%d¥n",prime[i]);
      printf("Total number of calculation : %d¥n",counter);
      return(0);
}
```

実行結果

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
Total number of calculation : 313
```

改良版2(prime3.c)

ソースファイル

```
#include <stdio.h>
int main(void)
{
    int   i,n;
    int   N=100; /* 最大の数 */
    int   prime[500]; /* 素数を格納する配列 */
    int   ptr=0; /* 抽出した素数の数 */
    int   counter=0; /* 乗算+除算の回数 */
    int   flag; /* 除算できたか否かのフラグ  */

    prime[ptr++]=2;   /* 最初の素数  */
    prime[ptr++]=3;   /* 2番目の素数  */

    for (n=5;n<=N;n+=2)     { /* 奇数のみを対象  */
        flag=0;
        for (i=1; counter++, prime[i]*prime[i]<=n;i++)    {
            counter++;
            if (n%prime[i]==0) { /* 割り切れたら素数でない  */
                flag=1;
                break;
            }
        }
        if (flag==0)
            prime[ptr++]=n;  /* 最後まで割り切れない */
    }
    for (i=0;i<ptr;i++) printf("%d¥n",prime[i]);
    printf("Total number of calculation : %d¥n",counter);
    return(0);
}
```

実行結果

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
Total number of calculation : 191
```


課題問題1(レポート)

課題の目的:素数を抽出する各種アルゴリズムを使用して計算量の概念を理解する

1. 各種アルゴリズムのプログラム作成
2. 各種アルゴリズムでN=10,50,100,500,1000として計算量を算出
3. 計算量をグラフ化する
4. 考察を記述する

★レポートを作成し次の講義(2015/5/7)に提出

★レポートのフォーマットは自由とする

再帰呼び出し(再帰処理)

☑再帰とは？

-もとにもどる、繰り返しの意味

☑プログラミングにおける意味

-ある関数の中で自分自身を呼び出す処理

-処理手順がそれ自身を用いて定義されている

-自分自身の呼び出しの終了条件が与えられている

-昔の言語ではサポートされていない(FORTRAN,BASIC)

☑効果

-繰り返し処理の代用が可能

-プログラムサイズを小さくできる(ソースコードレベル)

再帰呼び出しが適用できる例１

再帰処理基礎1

階乗：

1!=1

2!=2x1=2

3!=3x2x1=6

4!=

：

：

n!=n*(n-1)*(n-2)*….*2*1　　n!=n*(n-1)!

階乗が階乗の関数で表現できることに注目

数学的な決まり　0!=1　⇨　再帰終了条件

階乗を実現する関数例

（再帰呼び出しを使用しない）

```
int factorial(int n)
{
        int i,x;
        if (n==0)
        {
                return 1;
        }
        else
        {
                x=1;
                for (i=n;i>=1;i--)
                        {
                        x=x*i;  /* x *= i; */
                        }
                return x;
        }
}
```


階乗を実現する関数例

（再帰呼び出しを使用する）

```
int factorial(int n)
{
        if (n==0)
        {
                return 1;
        }
        else
        {
                return n*factorial(n-1);
        }
}
```

自明なケース

自明なケースの処理
再帰の終了条件

自分自身で定義される

プログラム比較

```
int factorial(int n)
{
        int i,x;
        if (n==0)
        {
                return 1;
        }
        else
        {
                x=1;
                for (i=n;i>=1;i--)
                        {
                        x=x*i;
                        }
                return x;
        }
}
```

```
int factorial(int n)
{
        if (n==0)
        {
                return 1;
        }
        else
        {
                return n*factorial(n-1);
        }
}
```

プログラムサイズに注目!!!
効果が理解できる

再帰呼び出しが適用できる例２

再帰処理基礎2

自然数の和：s(n)

s(1)=1

s(2)=1+2=3

s(3)=1+2+3=6

s(4)=

：

：

s(n)=1+2+3+・・・+(n-1)+n　　s(n)=s(n-1) + n

s(n)がs(n-1)の関数で表現できる

再帰：自分自身と同じ関数を呼び出す場合に記述可能

s(1)=1 ⇨ **再帰終了条件**

自然数の和を実現する関数例

（再帰呼び出しを使用しない）

```
int sum(int n)
{




}
```


自然数の和を実現する関数例

（再帰呼び出しを使用する）

```
int sum(int n)
{




}
```


演習問題2

課題：階乗と自然数の和を求める関数について、メイン関数を作成して動作を確認する。

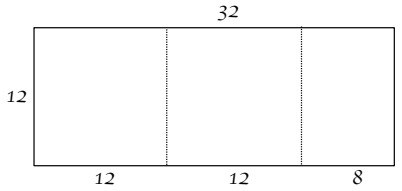
- 各関数のプログラム作成
- メイン関数の機能
  - キーボードから数値を入力する。
  - 結果を表示する。

再帰処理応用1

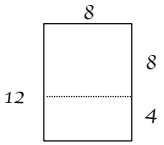
- ユークリッドの互除法：２つの自然数の最大公約数を求めるアルゴリズム
- 問題の置き換え：
  - 長方形を埋め尽くすことのできる正方形を考える
  - 埋め尽くすことが可能な最大の正方形の辺の長さを求める

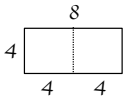
例題

- 32と12の最大公約数を求める
  - 長辺が32cm、短辺が12cmの長方形を分割



(1)長い辺を短い辺を1辺とする正方形で分割
(2)余った長方形に対して繰り返す
(3)最終的に分割された正方形の1辺が最大公約数





  - 1辺4cmの正方形で分割可能 ：最大公約数 4

手順の整理

２つの自然数(a≧b>0)が与えられる

1. aをbで割った余りをrとする。
2. rが０の場合　：　終了条件

   割った値が最大公約数
3. rが０以外

   a=b, b=rとして1.にもどる。

赤文字部分を再帰呼び出しで実現する

ユークリッド互除法　ソースコード

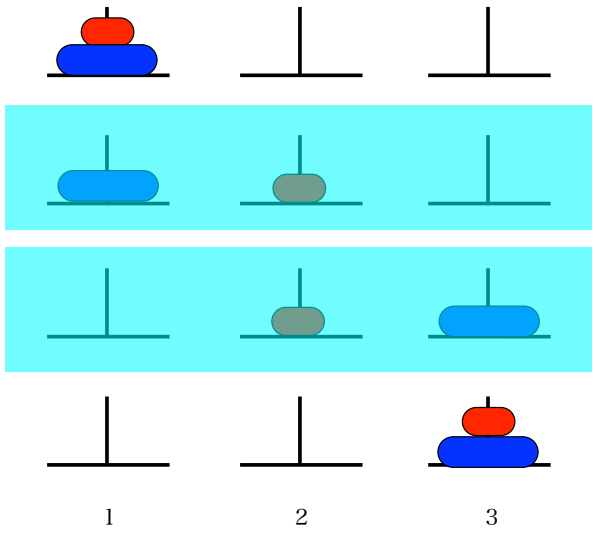
```
#include <stdio.h>

int gcd(int a, int b)
{
    if (b==0)
          return(a);
    else
          return(gcd(b,a % b));/*  再帰呼び出し  */
}

int main(void)
{
    int a,b;
    printf(" input a :");
    scanf("%d",&a);
    printf(" input b :");
    scanf("%d",&b);
    printf(" gcd is %d ¥n",gcd(a,b));

    return(0);
}
```

プログラムを実行して確認すること

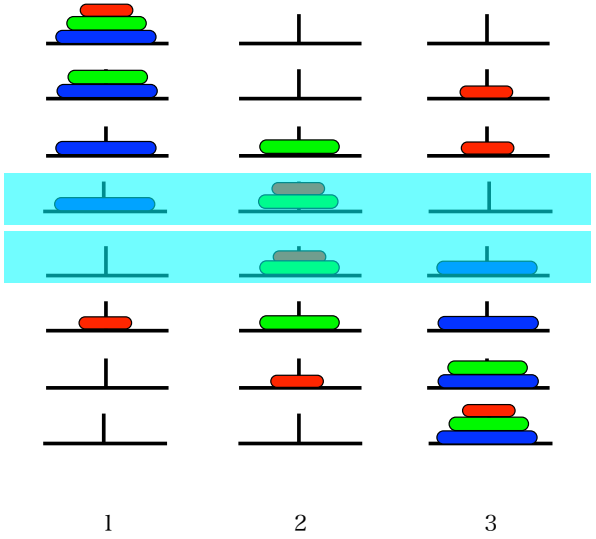
再帰処理応用２

- ハノイの塔：重なった円盤を3本の柱(軸1,2,3)の間で移動する問題
- 問題の規則：
  - 最初は軸1に全ての円盤が重ねられている
  - 円盤は1枚ずつのみ移動できる
  - 円盤を重ねる場合は、下の円盤よりも小さい円盤のみ重ねることが可能
  - 全ての円盤が軸３に移動したら終了

円盤が2枚の場合



円盤が３枚の場合



円盤の枚数によらない共通的な手順

1. 一番大きな円盤以外を軸２に移動
2. 一番大きな円盤を軸１から軸３に移動
3. 一番大きな円盤以外を軸３に移動

１および３を再帰的に実現する

ハノイの塔

ソースファイル

```
#include <stdio.h>


void move(int no, int x, int y)
{
   if (no > 1)
     move(no-1,x,6-x-y);
   printf("¥n[%d]%d=>%d",no,x,y);
   if (no > 1)
     move(no-1,6-x-y,y);
}


int main(void)
{
   int n;
   printf(" No of Disc :");
   scanf("%d",&n);
   move(n,1,3);
   return(0);
}
```

実行結果

No of Disc :2

```
[1] 1 => 2
[2] 1 => 3
[1] 2 => 3
```

No of Disc :3

```
[1] 1 => 3
[2] 1 => 2
[1] 3 => 2
[3] 1 => 3
[1] 2 => 1
[2] 2 => 3
[1] 1 => 3
```

No of Disc :4

```
[1] 1 => 2
[2] 1 => 3
[1] 2 => 3
[3] 1 => 2
[1] 3 => 1
[2] 3 => 2
[1] 1 => 2
[4] 1 => 3
[1] 2 => 3
[2] 2 => 1
[1] 3 => 1
[3] 2 => 3
[1] 1 => 2
[2] 1 => 3
[1] 2 => 3
```

プログラムを実行して確認すること