平成 24 年度 アドバンスト・プログラミング
(Advanced Programming)

小澤　一文，陳　国躍，中村真輔

木曜 2 限 GI201 教室

再帰呼び出しの高速化

フィボナッチ数列のプログラム

```
#include<stdio.h>
int fib(int n);
int f[45];
main() {
  int n;
  f[0]=f[1]=1;
  for (n=2; n<45; n++) f[n]=0;

  for (n=0; n<45; n++)
    printf(" f(%2d)=%10d \n",n,fib(n));
}
int fib(int n) {
  if (f[n] == 0)
    f[n]=fib(n-1)+fib(n-2);
  return f[n];
}
```

- main 関数でも fib 関数でも使える配列 `f[]` を定義する
- 計算結果を配列 `f[]` に記憶させておく
- 計算済みのときは配列 `f[]` から値を拾って返すだけ

変数の種類

1. 自動変数
   通常使われる変数。関数が呼び出されるたびに記憶領域が生成される (初期化される)

2. 静的変数
   return で一旦抜けだし戻ってきた後も値が保存されている

3. 大域変数
   プログラムの冒頭で宣言された変数で，すべての関数に共通

4. レジスタ変数。
   自動変数の一つでありが，記憶領域は主記憶でなくレジスタ上である。頻繁にアクセする変数向き。

5. 外部変数
   他のプログラムで宣言された変数。

各変数の違い

```
#include<stdio.h>
void func();
int x=0; /*  大域変数  */
main () {
  int i;
  for (i=0; i<5; i++) {
    x++;
    func();
  }
}
void func() {
  int a=0; /*  自動変数  */
  static int b=0; /*  静的変数  */
  a++;
  b++;
  printf(" a = %d b = %d x= %d\n",a,b,x);
}
```

実行例

```
 a = 1 b = 1 x= 1
 a = 1 b = 2 x= 2
 a = 1 b = 3 x= 3
 a = 1 b = 4 x= 4
 a = 1 b = 5 x= 5
```

外部変数

```
/*  extern.c  */
int a=100;  /*  外部変数  */
int b=200;  /*  外部変数  */

/*  main.c  */
#include<stdio.h>
void func();
main() {
  extern int a,b;
  printf("a+b = %d \n",a+b);
  func();
}
void func() {
  extern int a,b;
  printf("a*b = %d \n",a*b);
}
```

実行例

```
gcc extern.c main.c
./a.out
a+b = 300
a*b = 20000
```

レジスタ変数

```
#include <stdio.h>
main() {
   register int i;
   int j=4;
   for (i=0; i<10; i++) j=2*j+i;
   printf(" j = %d \n",j);
}
```

実行例

```
 j = 5109
```

レジスタ変数はコンパイラが最適化処理をしてくれるのであまり必要がない

ビット演算

C 言語では以下にに示すようなビット単位の演算子が用意されている

1. `&` : `a & b` で変数 `a, b` の bit 毎の and を計算する。
2. `|` : `a | b` で変数 `a, b` の bit 毎の or を計算する。
3. `~` : `~a` で変数 `a` の各 bit を反転させる (not)。
4. `^` : `a ^ b` で変数 `a, b` の bit 毎の xor を計算する。
5. `>>`: `a >> n` で変数 `a` を n bit 算術右シフトする。
6. `<<`: `a << n` で変数 `a` を n bit 算術左シフトする。

2 進表示

整数型変数をビット列で表示する関数

```
void Disp_b(int x) {
  int i,bit[32];
  unsigned int w;

  w=x;
  for (i=0; i<=31; i++) {
    bit[i]=w & 1;    /*  最下位の 1bit を取り出す   */
    w>>=1;  /*  w を 1bit 右シフトする   */
  }
  for (i=31; i>=0; i--) {
    printf("%d",bit[i]);
    if ((i % 4) == 0) printf(" ");
  }
}
```

16 進表示

整数型変数を 16 進数で表示する関数

```
void Disp_h(int x) {
  int i, Hex[8];
  unsigned int w;
  char h[16]={'0','1','2','3','4','5','6','7','8',
      '9','A','B','C','D','E','F'};
  w=x;
  for (i=0; i<=7; i++) {
    Hex[i]=w & 15;  /*  最下位の 4bit を取り出す   */
    w>>=4;  /*  w を 4bit 右シフトする   */
  }
  for (i=7; i>=0; i--)
    printf("%c",h[Hex[i]]);
}
```

実行例 (1)

```
main() {
   int i;
   scanf("%d",&i);
   printf(" %d = ",i);
   Disp_b(i); printf("\n");
}
```

実行例

```
 1349871 = 0000 0000 0001 0100 1001 1000 1110 1111
```

```
main() {
   int i;
   scanf("%d",&i);
   printf(" %d = ",i);
   Disp_h(i); printf("\n");
}
```

実行例

```
-482522 = FFF8A326
```

実行例 (2)

```
main() {
  int a=0xfedcba98, b=0x1234abcd, x,y,z,u,v;
  printf("   a= %x,   b=%x \n",a,b);
  printf("   a    = "); Disp_b(a); printf("\n");
  printf("   b    = "); Disp_b(b); printf("\n");
  printf("\n");

  x=a & b; printf(" a & b  = "); Disp_b(x); printf("\n");
  y=a | b; printf(" a | b  = "); Disp_b(y); printf("\n");
  z=~a;  printf("  ~a    = "); Disp_b(z); printf("\n");
  u=~b;  printf("  ~b    = "); Disp_b(u); printf("\n");
  v=a^b; printf(" a ^ b  = "); Disp_b(v); printf("\n");
}
```

実行結果

```
   a= fedcba98,   b=1234abcd
   a    = 1111 1110 1101 1100 1011 1010 1001 1000
   b    = 0001 0010 0011 0100 1010 1011 1100 1101

 a & b  = 0001 0010 0001 0100 1010 1010 1000 1000
 a | b  = 1111 1110 1111 1100 1011 1011 1101 1101
  ~a    = 0000 0001 0010 0011 0100 0101 0110 0111
  ~b    = 1110 1101 1100 1011 0101 0100 0011 0010
 a ^ b  = 1110 1100 1110 1000 0001 0001 0101 0101
```

演習問題

1. 2 つの整数型変数を bit 毎に比較し，異なる部分の個数を数えるプログラムを書け。

2. 整数型変数 `i` に `~i+1` という演算を施すと結果はどのようになるか。またそうなる理由を考察せよ。

3. bit 操作のみで次のような模様を描くプログラムを作れ。

```
1111 0000 1111 0000 1111 0000 1111 0000
0000 1111 0000 1111 0000 1111 0000 1111
1111 0000 1111 0000 1111 0000 1111 0000
0000 1111 0000 1111 0000 1111 0000 1111
1111 0000 1111 0000 1111 0000 1111 0000
0000 1111 0000 1111 0000 1111 0000 1111
1111 0000 1111 0000 1111 0000 1111 0000
0000 1111 0000 1111 0000 1111 0000 1111
```