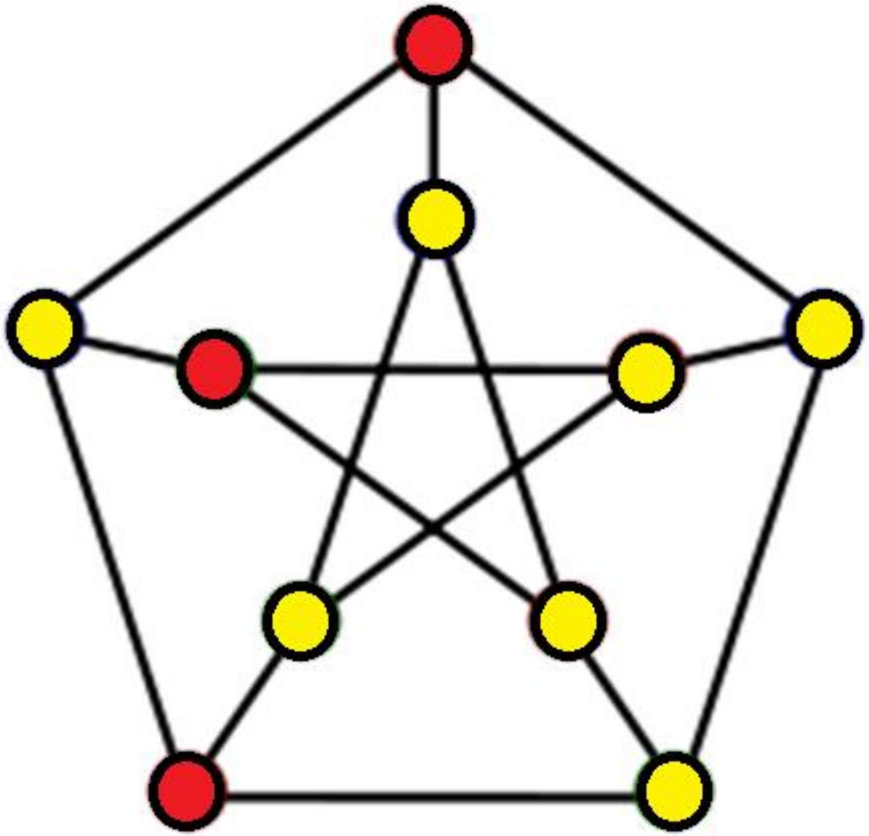
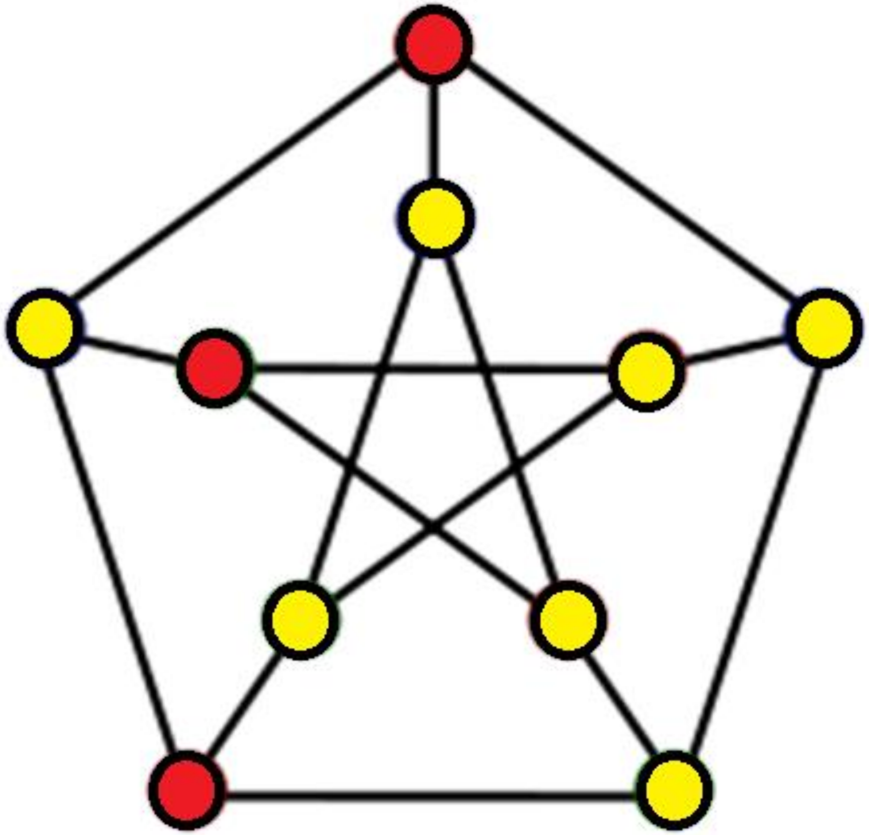
ref_value.c: What does this print?

```c
#include <stdio.h>
#include <stdlib.h>
int dumb_function(int *b, int c);
main()
{   int x, y, z;
    x=1; y=2; z= 3;
    x= dumb_function(&y, z);
    printf("x= %3d   y=%3d   z= %3d\n", x, y, z);
}
int dumb_function(int *b, int c)
{  int a;
   a= 4;   *b= 5;  c= 6;
   return(a);
}
```

A *dominating set* of a graph G is a subset D of the vertices of G such that every vertex v of G is either in the set D or v has at least one neighbour that is in D.



A *dominating set* of a graph G is a subset D of the vertices of G such that every vertex v of G is either in the set D or v has at least one neighbour that is in D.



Simple degree bound: if the maximum degree of a vertex in the graphs is Δ then the minimum dominating set size is at least n/(Δ +1).

Note: I sometimes have to reformat and delete comments/error handling to get code blocks all on one screen at a viewable font size.

Please use:

- meaningful variable names
- nice indentation style
- lots of comments
- error detection and handling

in your programs.

Nicer formating

```
#include <stdio.h>
#include <stdlib.h>
int dumb_function(int *b, int c);
main()
{
        int x, y, z;

//      Examples of call by reference and call by value.

        x=1;
        y=2;
        z= 3;

        x= dumb_function(&y, z);

        printf("x= %3d   y=%3d   z= %3d\n", x, y, z);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#define NMAX 256
#define DEBUG 1

int   read_graph(int *n, int G[NMAX][NMAX]);
void print_graph(int n,  int G[NMAX][NMAX]);

main()
{
    int n;
    int G[NMAX][NMAX];
    int n_graph;
```

```
    n_graph=0;

    while (read_graph(&n, G))
    {
        n_graph++;

#if DEBUG
        printf("Input graph %3d:\n", n_graph);
        print_graph(n, G);
#endif

        // Read certificate then check it here.


    }
}
```

Sample input file: in.txt

```
4
3  1  2  3
2  0  3
2  0  3
3  1  2  3

6
2  5  1
2  0  2
2  1   3
2  2  4
2  3  5
2  4  0
```

```
int read_graph(int *n, int G[NMAX][NMAX])
{
    int i, j, u, d;

    if (scanf("%d", n)!=1) return(0);

  // IMPORTANT: some compilers initialize
  // variables but others do not. Be safe and
  // initialize everything always!!!

    for (i=0; i < *n; i++)
      for (j=0; j < *n; j++)
        G[i][j]=0;
```

```
// Error handling not included!
    for (i=0; i < *n; i++)
    {
        if (scanf("%d", &d) != 1) exit(0);  //  degree
        for (j=0; j < d; j++)
        {
            if (scanf("%d", &u)!= 1)
            {
                exit(0); // Error!
            }
            G[i][u]= 1;
        }
    }
    return(1);
}
```

```
void print_graph(int n, int G[NMAX][NMAX])
{       int i, j, d;

        for (i=0; i < n; i++)
        {
              d=0;
              for (j=0; j < n; j++)
                   d+= G[i][j];
              printf("%3d (%1d):", i, d);
              for (j=0; j < n; j++)
              {
                    if (G[i][j]) printf(" %3d", j);
              }
              printf("\n");
        }
}
```

IMPORTANT POINTS:

I read from standard input and write to standard output. But that does NOT mean we cannot use files for the I/O.

```
gcc *.c
```

```
mv a.out dom_set
```

```
time dom_set < in.txt > out.txt
```

DOS: makes a.exe instead of a.out.

Output file out.txt:

```
Input graph   1:
  0 (3):   1   2   3
  1 (2):   0   3
  2 (2):   0   3
  3 (3):   1   2   3
Input graph   2:
  0 (2):   1   5
  1 (2):   0   2
  2 (2):   1   3
  3 (2):   2   4
  4 (2):   3   5
  5 (2):   0   4
```

```
#include <stdio.h>
#include <stdlib.h>
#define NMAX 256
#define DEBUG 1
void  gen_hypercube(int d, int *n,
                       int G[NMAX][NMAX]);
void print_graph(int n, int G[NMAX][NMAX]);
main(int argc, char *argv[])
{

   int n;
   int G[NMAX][NMAX];
   int min_dim, max_dim;
   int dim;
```

Difference from JAVA:

argv[0] is name such as a.out used when invoking the program and argv[1], argv[2] … have the other values you type in represented as strings.
atoi: converts an ascii string to an integer value.

```
if (argc != 3)
{
  printf("Usage: %s <min_dim><max_dim>\n",
      argv[0]);
  exit(0);
}
min_dim= atoi(argv[1]);
max_dim= atoi(argv[2]);
```

```
for (dim= min_dim; dim <= max_dim; dim++)
{
    gen_hypercube(dim, &n, G);
#if DEBUG
    printf(
    "Hypercube of dimension %2d: %2d vertices\n",
       dim, n);
#endif
#if 0
         print_graph(n, G);
#endif


   }
}
```

```
void  gen_hypercube(int d, int *n,
                             int G[NMAX][NMAX])
{
      *n= 1 << d;

       // Add code here.


}
```

```
gcc *.c
```

```
mv a.out gen
```

```
gen
```

The program prints:

```
Usage: gen <min_dim><max_dim>
```

```
gen 3 6
```

The program prints:

```
Hypercube of dimension  3:  8 vertices
Hypercube of dimension  4: 16 vertices
Hypercube of dimension  5: 32 vertices
Hypercube of dimension  6: 64 vertices
```

When the hypercube generator is debugged and we want to compute the dominating sets:

```
gen 3 6 | dom_set > ohypercube_3_6.txt
```

or if your path is not set up nicely:

```
./gen 3 6 | ./dom_set > ohypercube_3_6.txt
```

This strategy eliminates the need to store the generated graphs in files. Only the output is recorded.

This is very advantageous if you want to generate and test a very large set of graphs.

Another example of code using a command line parameter:

```
int main(int argc, char *argv[])
{
  int verbose;

  if (argc!= 2)
  {
      printf("Usage: %s <verbose>\n", argv[0]);
      exit(0);
   }
   verbose= atoi(argv[1]);
```