A.

```c
#include<stdio.h>
int main()
{
    printf("Hello World\n");
    return 0;
}
```

B.

```c
#include<stdio.h>
int main()
{
    int a,b;
    scanf("%d %d",&a,&b);
    printf("%d",a+b);
    return 0;
}
```

C.

```c
#include <stdio.h>
int main()
{
    int year;
    scanf("%d", &year);
    if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))
    {
        printf("Yes\n");
    }
    else
    {
        printf("No\n");
    }
    return 0;
}
```

D.

```c
#include <stdio.h>
int main() {
    int num;
    scanf("%d", &num);
    for (int i = 1; i <= num; i++) {
        if (num % i == 0) {
            printf("%d\n", i);
        }
    }
  return 0;
}
```

E.

```c
#include <stdio.h>

int countSteps(long long n)
{
    long long steps = 0;
    while (n != 1)
    {
        if (n % 2 == 0)
        {
            n = n / 2;
        }
        else
        {
            n = n + 1;
        }
        steps++;
    }

    return steps;
}
```

```
int main()
{
    long long n;
    scanf("%lld", &n);

    printf("%lld\n", countSteps(n));

    return 0;
}
```

F.
Your output may not match sample output exactly but still can be valid.

```
#include <stdio.h>
int main()
{
    int T;
    scanf("%d", &T);

    while (T--)
    {
        int n, a=0;
        scanf("%d", &n);
        if (n > 10)
        {
            a = 10;
        }
        printf("%d %d\n", a, n - a);
    }
    return 0;
}
```

G.
Here we use the concept of prefix sum to optimize the solution. We reduced the operation of iterating over the array to only perform a single subtraction to get the sum of range for each query.

```c
#include <stdio.h>
int main()
{
    long long int n,q;
    scanf("%lld %lld",&n, &q);
    long long int arr[n];
    long long int psum[n+1];
    psum[0] = 0;
    for (long long int i = 0; i < n; i++)
    {
        scanf("%lld",&arr[i]);
        psum[i+1] = psum[i] + arr[i];
    }
    while (q--)
    {
        long long int l,r;
        scanf("%lld %lld",&l, &r);
        printf("%lld\n",psum[r] - psum[l-1]);
    }
    return 0;
}
```

H.
We used the formula given in the question to get the value of PI.

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

int main()
{
    int T;
    double r;
    scanf("%d", &T);

    for (int i = 1; i <= T; i++)
    {
        scanf("%lf", &r);

        double square_area = 4 * r * r;

        double circle_area = 2 * acos(0.0) * r * r;

        double shaded_area = square_area - circle_area;

        printf("Case %d: %.2lf\n", i, shaded_area);
    }

    return 0;
}
```

I:- This problem was for finding extra talented persons. You will probably not understand the solution right now. Take some time, You will learn this things in a few months if you stay consistent.

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

void swap(char str[], int i, int j)
{
    char temp = str[i];
    str[i] = str[j];
    str[j] = temp;
}

int next_permutation(char str[])
{
    int len = strlen(str);
    int i;
    for (i = len - 2; i >= 0; i--)
    {
        if (str[i] < str[i + 1])
        {
            break;
        }
    }
    if (i < 0)
        return 0;
    int j;
    for (j = len - 1; j > i; j--)
    {
        if (str[j] > str[i])
        {
            break;
        }
    }
```

```
    swap(str, i, j);

    int left = i + 1;
    int right = len - 1;
    while (left < right)
    {
        swap(str, left, right);
        left++;
        right--;
    }

    return 1;
}

int main()
{
    int T;
    scanf("%d", &T);

    for (int t = 1; t <= T; t++)
    {
        int N, K;
        scanf("%d %d", &N, &K);
        char str[27] = "";
        for (int i = 0; i < N; i++)
        {
            str[i] = 'A' + i;
        }
        str[N] = '\0';

        printf("Case %d:\n", t);

        printf("%s\n", str);
        K--;
        while (K > 0 && next_permutation(str))
```

```
        {
            printf("%s\n", str);
            K--;
        }
    }

    return 0;
}
```

J. Same as last problem. After learning basic DP, you will get this.

```
#include <stdio.h>
int max(int a, int b) {
    return (a > b) ? a : b;
}
int can_break(int n, int a, int b, int c, int dp[]) {

    if(n == 0) return 1;
    if(n < 0) return 0;
    if(dp[n] != -1) return dp[n];
    dp[n] = can_break(n-a, a, b, c, dp) ||
            can_break(n-b, a, b, c, dp) ||
            can_break(n-c, a, b, c, dp);

    return dp[n];
}

int max_pieces(int n, int a, int b, int c) {
    int dp[4001];
    for(int i = 0; i <= n; i++) {
        dp[i] = -1;
    }
    if(!can_break(n, a, b, c, dp)) {
        return 0;
    }
```

```
    dp[0] = 0;
    for(int i = 1; i <= n; i++) {
        dp[i] = -1;

        if(i >= a && dp[i-a] != -1) {
            dp[i] = dp[i-a] + 1;
        }
        if(i >= b && dp[i-b] != -1) {
            dp[i] = max(dp[i], dp[i-b] + 1);
        }
        if(i >= c && dp[i-c] != -1) {
            dp[i] = max(dp[i], dp[i-c] + 1);
        }
    }

    return dp[n];
}

int main() {
    int n, a, b, c;
    scanf("%d %d %d %d", &n, &a, &b, &c);

    printf("%d\n", max_pieces(n, a, b, c));

    return 0;
}
```