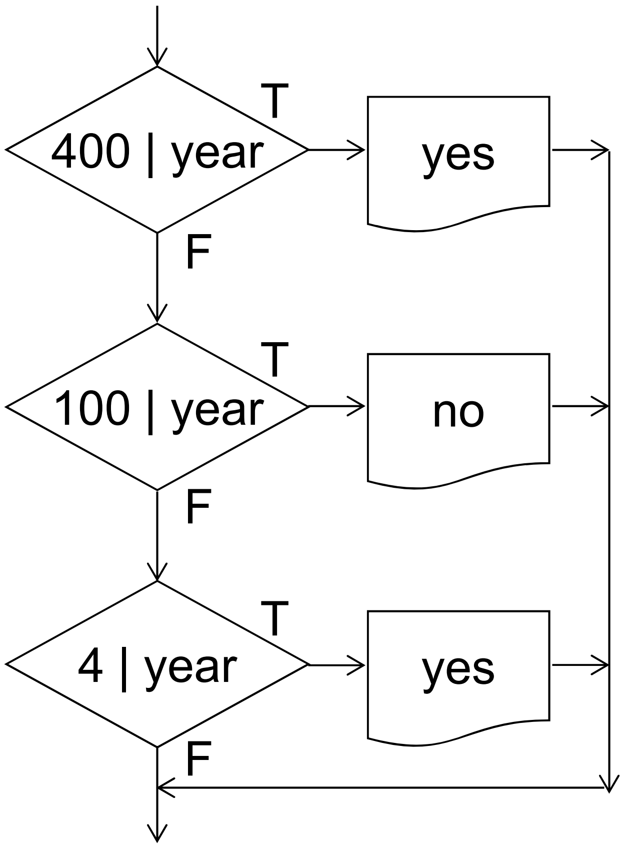
More Examples w/o Using Arrays

Pei-yih Ting

Check Leap Years

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int year;
    printf("Enter a year to check if it is a leap year\n");
    scanf("%d", &year);
    if ( year%400 == 0)
        printf("%d is a leap year.\n", year);
    else if ( year%100 == 0)
        printf("%d is not a leap year.\n", year);
    else if ( year%4 == 0 )
        printf("%d is a leap year.\n", year);
    else
        printf("%d is not a leap year.\n", year);
    system("pause");
    return 0;
}
```



Check Vowels

- **Version 1**

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    char ch;
    printf("Enter a character: ");
    scanf("%c", &ch);
    if (ch == 'a' || ch == 'A' || ch == 'e' || ch == 'E' ||
        ch == 'i' || ch == 'I' || ch =='o' || ch=='O' ||
        ch == 'u' || ch == 'U')
        printf("%c is a vowel.\n", ch);
    else
        printf("%c is not a vowel.\n", ch);
    system("pause");
    return 0;
}
```

Check Vowels (cont'd)

- **Version 2**

```
switch (ch) {
 case 'a': case 'A':
 case 'e': case 'E':
 case 'i':  case 'I':
 case 'o': case 'O':
 case 'u': case 'U':
     printf("%c is a vowel.\n", ch);
     break;
 default:
     printf("%c is not a vowel.\n", ch);
 }
```

Check Vowels (cont'd)

- **Version 3**

```
if (check_vowel(ch))
    printf("%c is a vowel.\n", ch);
else
    printf("%c is not a vowel.\n", ch);

int check_vowel(char ch) {
  if (ch >= 'A' && ch <= 'Z')
      ch = ch - 'A' + 'a';
  if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')
      return 1;
  return 0;
}
```

Check Vowels (cont'd)

- **Version 4 (with array)**

```
int check_vowel(char ch) {
    char vowels[] = "aeiouAEIOU";
    int i;
    for (i=0; i<10; i++)
        if (ch == vowels[i])
            return 1;
    return 0;
}
```

```
*(vowels+i)
```

```
sizeof(vowels)-1
strlen(vowels)
```

```
int check_vowel(char ch) {
    char *vowels = "aeiouAEIOU";
    int i;
    for (i=0; i<10; i++)
        if (ch == vowels[i])
            return 1;
    return 0;
}
```

```
int check_vowel(char ch) {
    int i;
    for (i=0; i<10; i++)
        if (ch == "aeiouAEIOU"[i])
            return 1;
    return 0;
}
```

Test multiples of 11

- 112233 is a multiple of 11 because
  -1+1-2+2-3+3 = 0, 0 is a multiple of 11
- 3234556931234861 is not a multiple of 11 because
  -3+2-3+4-5+5-6+9-3+1-2+3-4+8-6+1 = 1 is not a multiple of 11
- 9058698 is a multiple of 11, -9+0-5+8-6+9-8=-11
- Note: $a_n 10^n + a_{n-1} 10^{n-1} + a_{n-2} 10^{n-2} + \ldots + a_0 \equiv$
  $a_n (11-1)^n + a_{n-1} (11-1)^{n-1} + a_{n-2} (11-1)^{n-2} + \ldots + a_0 \equiv$
  $11(\ldots) + (-1)^n a_n + (-1)^{n-1} a_{n-1} + (-1)^{n-2} a_{n-2} + \ldots + a_0 \equiv$
  $(-1)^n a_n + (-1)^{n-1} a_{n-1} + (-1)^{n-2} a_{n-2} + \ldots + a_0 \pmod{11}$
- Please use a loop to read the input character by character Note: if the input has correct format, you can use scanf("%c", …) or scanf("%1d", …)
- You should process each character on the fly w/o using arrays

Prime Testing

- Trial and error

```
for (i=2; i<n-1; i++)
    if (n%i==0) {
        printf("%d | %d\n", i, n);
        break;
    }
```

- Trial and error w/ range

```
for (i=2; i<=n/2; i++)
    if (n%i==0) {
        printf("%d | %d\n", i, n);
        break;
    }
```

```
for (i=2; i<=sqrt(n); i++)
    if (n%i==0) {
        printf("%d | %d\n", i, n);
        break;
    }
```

Approximating π

- The value for π can be determined by the series equation

  $\pi = 4 \times (1 – 1/3 + 1/5 – 1/7 + 1/9 – 1/11 + 1/13 - \ldots)$

- Write a program to approximate the value of π using the given formula including terms up through 1/**99**
- Write a program to approximate the value of π using the given formula until the variation caused by the last term is less than **0.1%** of the approximated π value

Calculating x^n mod p

- Let x, p be two integers, and n be a big integer, calculate **x^n mod p** (e.g. $12^{1000000000}$ mod 123457) efficiently
- If we write $n = b_k \cdot 2^k + b_{k-1} \cdot 2^{k-1} + \ldots + b_0$ (i.e. binary representation)

then $x^n = x^{(((\ldots(b_k \cdot 2+b_{k-1})\cdot 2+\ldots+ b_3)\cdot 2+b_2)\cdot 2+b_1)\cdot 2+b_0}$

$= (((\cdots((x^{b_k})^2 x^{b_{k-1}})^2 \cdots x^{b_3})^2 x^{b_2})^2 x^{b_1})^2 x^{b_0}$ method 1

or $x^n = x^{b_0+2(b_1+2(b_2+2(b_3+\ldots+2(b_{k-1}+2b_k)\ldots)))}$

$= x^{b_0} \cdot (x^{2^1})^{b_1} \cdot (x^{2^2})^{b_2} \cdot (x^{2^3})^{b_3} \ldots (x^{2^{k-1}})^{b_{k-1}} \cdot (x^{2^k})^{b_k}$

method 2

由 $b_0, b_1, \ldots$ 算到 b_k 比較簡單

- 1st step: derive $b_0, b_1, \ldots, b_k$ from n

```
q = n;
while (q>0) {
    b = q % 2;
    q = q / 2;
}
```

- 2nd step: x^{2^k}

```
q = n; xp = x;
while (q>0) {
    b = q % 2;
    q = q / 2;
    xp = (xp*xp)%p;
}
```

- 3rd step: $(x^{2^k})^{b_k}$

```
q = n; xp = x; r = 1;
while (q>0) {
    if (q%2) r = (r*xp)%p;
    q = q / 2;
    xp = (xp*xp)%p;
}
```

Calculating x^n mod p

- Let x, p be two integers, and n be a big integer, calculate **x^n mod p** (e.g. $12^{1000000000}$ mod 123457) efficiently
- If we write $n = b_k \cdot 2^k + b_{k-1} \cdot 2^{k-1} + \ldots + b_0$ (i.e. binary representation)

then $x^n = x^{(((\ldots(b_k \cdot 2+b_{k-1})\cdot 2+\ldots+ b_3)\cdot 2+b_2)\cdot 2+b_1)\cdot 2+b_0}$

$= (((\cdots((x^{b_k})^2 x^{b_{k-1}})^2 \cdots x^{b_3})^2 x^{b_2})^2 x^{b_1})^2 x^{b_0}$ method 1

or

$x^n = x^{b_0+2(b_1+2(b_2+2(b_3+\ldots+2(b_{k-1}+2b_k)\ldots)))}$

$= x^{b_0} \cdot (x^{2^1})^{b_1} \cdot (x^{2^2})^{b_2} \cdot (x^{2^3})^{b_3} \ldots (x^{2^{k-1}})^{b_{k-1}} \cdot (x^{2^k})^{b_k}$

method 2

由 $b_0, b_1, \ldots$ 算到 b_k 比較簡單

- 1st step: derive $b_0, b_1, \ldots, b_k$ from n

```
q = n;
while (q>0) {
    b = q % 2;
    q = q / 2;
}
```

- 2nd step: x^{2^k}

```
q = n; xp = x;
while (q>0) {
    b = q % 2;
    q = q / 2;
    xp = (xp*xp)%p;
}
```

- 3rd step: $(x^{2^k})^{b_k}$

```
q = n; xp = x; r = 1;
while (q>0) {
    if (q%2) r = (r*xp)%p;
    q = q / 2;
    xp = (xp*xp)%p;
}
```

Calculate C^n_k and P^n_k

- **Version 1**

$$\frac{n!}{k!\,(n-k)!}$$

```
#include <stdio.h>
#include <stdlib.h>
int factorial(int);
int find_nCk(int, int);
int find_nPk(int, int);
int main(void) {
    int n, k, nCk, nPk;
    printf("Enter the value of n and k\n");
    scanf("%d%d",&n,&k);
    nCk = find_nCk(n, k);
    nPk = find_nPk(n, k);
    printf("%dC%d = %d\n", n, k, nCk);
    printf("%dP%d = %d\n", n, k, nPk);
    system("pause");
    return 0;
}
int find_nCk(int n, int k) {
    return factorial(n) / (factorial(k)*factorial(n-k));
}
int find_nPk(int n, int k) {
    return factorial(n)/factorial(n-k);
}
int factorial(int n) {
    int i, result=1;
    for (i=1; i<=n; i++)
        result = result*i;
    return result;
}
```

$$\frac{n!}{(n-k)!}$$

$$P^{15}_3 = 15 \cdot 14 \cdot 13 = 2730$$

$15! = 1307674368000$

$2^{31}-1 = 2147483647$

Conceptually correct, actually has serious problem!!

Calculate C^n_k and P^n_k

- **Version 2** **long long int**

```c
#include <stdio.h>
#include <stdlib.h>
long long int factorial(int);
int find_nCk(int, int);
int find_nPk(int, int);
int main(void) {
    int n, k, nCk, nPk;
    printf("Enter the value of n and k\n");
    scanf("%d%d",&n,&k);
    nCk = find_nCk(n, k);
    nPk = find_nPk(n, k);
    printf("%dC%d = %d\n", n, k, nCk);
    printf("%dP%d = %d\n", n, k, nPk);
    system("pause");
    return 0;
}
int find_nCk(int n, int k) {
    return factorial(n)/(factorial(k)*factorial(n-k));
}
int find_nPk(int n, int k) {
    return factorial(n)/factorial(n-k);
}
long long int factorial(int n) {
    int i; long long int result=1;
    for (i=1; i<=n; i++)
        result = result*i;
    return result;
}
```

$$P^{21}_{3} = 21 \cdot 20 \cdot 19 = 7980$$

Better! But still quite limited.

21! = 51090942171709440000

2^{63}-1= 9223372036854775807

Calculate C_k^n and P_k^n

- **Version 3** **skip unnecessary factorial n!, (n-k)!**

$$P_k^n = n\,(n-1) \cdots (n-k+1)$$

```
int find_nPk(int n, int k) {
    int i, result=1;
    for (i=0; i<k; i++)
        result = result * (n-i);
    return result;
}
```

Ex. 30P6 = 427518000
30P7 = 1670497408
30P8 = -233265280
100P5 = 444567808
100P6 = -715731200

```
long long int find_nPk(int n, int k) {
    int i;
    long long int result=1;
    for (i=0; i<k; i++)
        result = result * (n-i);
    return result;
}
```

Ex. 30P13 = 7457470769548800000
30P14 = -5769043765476591616
100P10 = 7475418734400817152
100P11 = 8704899442529685504
100P12 = -27200710659158016

```
printf("result = %I64d\n", result);
printf("result = %lld\n", result);
```

Calculate C^n_k and P^n_k

- **Version 3 (cont'd)**

```
long long find_nCk(int n, int k)  {
    int i;
    long long int numerator=1, denominator=1;
    for (i=0; i<k; i++) {
        numerator *= n-i;
        denominator *= k-i;
    }
    return numerator/denominator;
}
```

30C14 = -66175233
100C10 = 2060025003968

```
long long int find_nCk(int n, int k) {
    int i;
    double result=1.0;
    for (i=0; i<k; i++)
        result *= ((double)(n-i))/(k-i);
    return (long long int) (result+0.5);
}
```

100C18 = -9223372036854775808

$$C^n_k = \frac{n\,(n-1)\cdots(n-k+1)}{k\,(k-1)\cdots 1}$$

```
long long find_nCk(int n, int k) {
    int i;
    long long numerator=1;
    long long denominator=1;
    for (i=0; i<k; i++) {
        numerator *= n-i;
        if (numerator % (k-i) == 0)
            numerator /= k-i;
        else
            denominator *= k-i;
    }
    return numerator/denominator;
}
```

100C15 = 3327654233778599

Calculating e^x

- Approximate e^x with the formula

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \ldots \text{ until } \quad \frac{x^n}{n!} < 10^{-5}$$

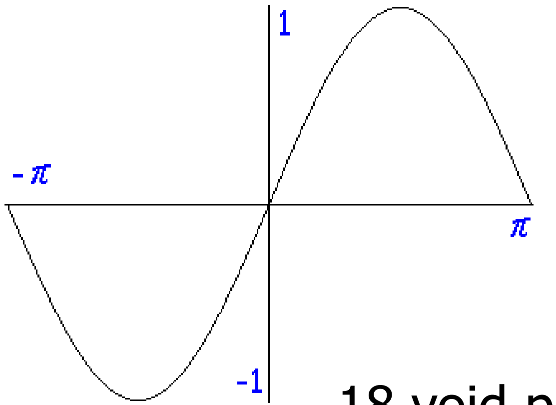
- Do not calculate factorial n! or x^n alone again!
- This is a number approximately 2.71828^x, which is larger than 2^x. If x is large, it will be a huge number. However, when you calculate n! alone, either it will overflow if you use *int* type; or the representation error is high if you use *double* type.
- It should be quite natural to use *double* type in the calculation of each term of the series.

Series approximation of e^x

```
#include <stdio.h>

void main()
{
    double x, sum, current;
    int i;
    printf("Please input an exponent: ");
    scanf("%lf", &x);
    sum = 1 + x; i = 2; current = x * x / i;
    while (current/sum > 0.0001)
    {
        sum += current;
        i++;
        current = current * x / i;
    }
    printf("e^%lf=%lf\n", x, sum);
}
```

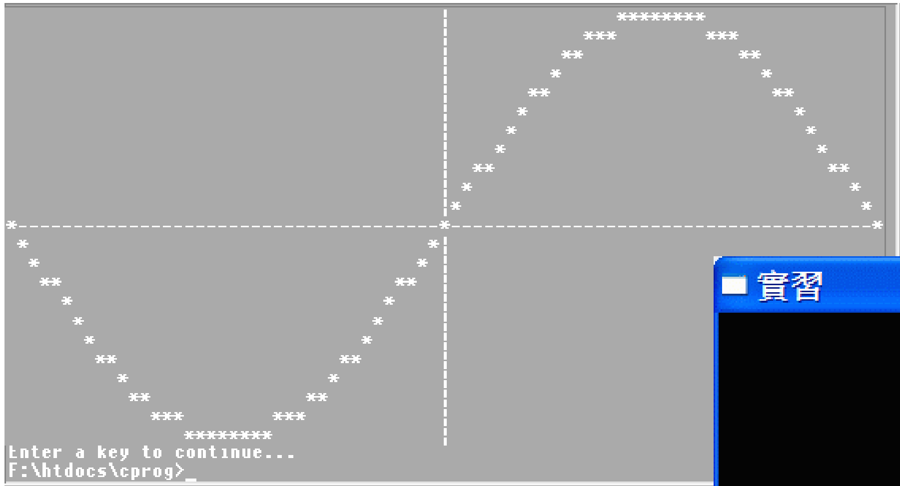
Plot Sine Function in Text Mode

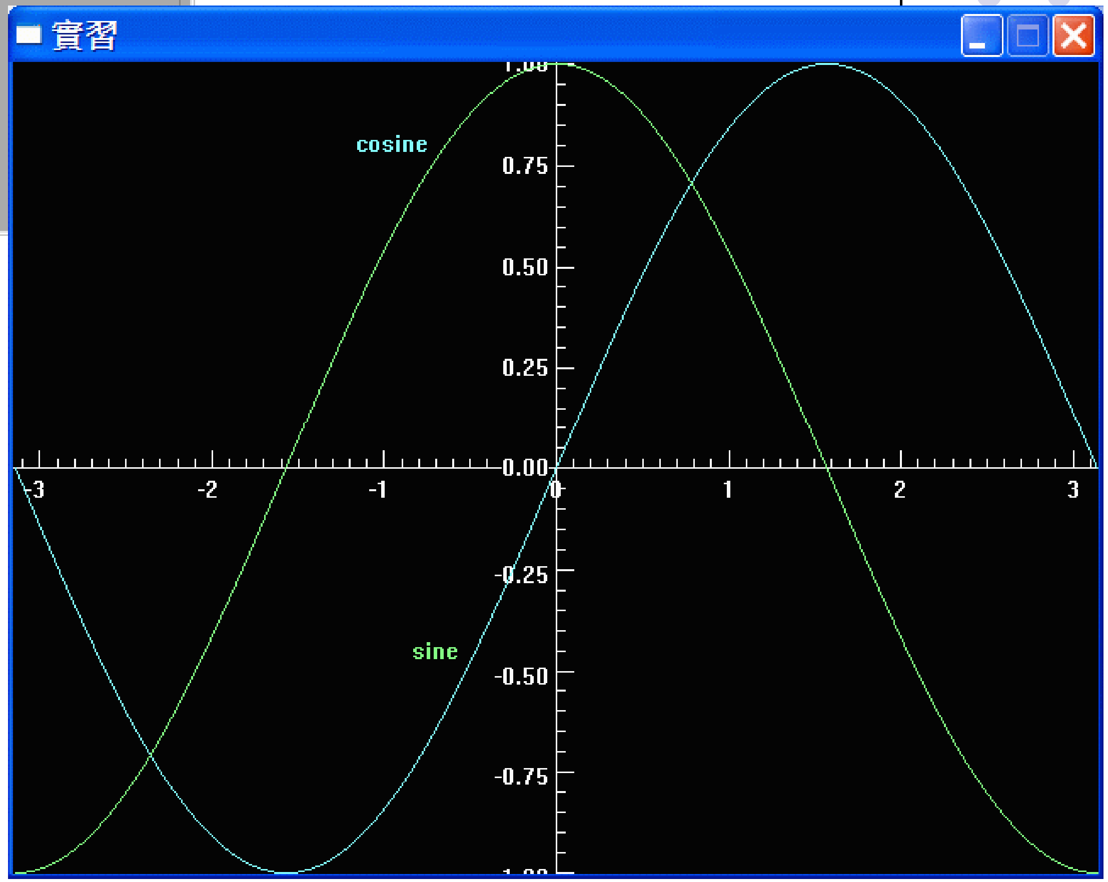



```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void printStar(int position);

int main(void)  {
    int i;
    int iy;
    const double pi = atan(1.0)*4;
    for (i=0; i<30; i++) {
        iy = (sin(i/30.0*2*pi)+1) * (30-1e-10) + 1; // [1.0, 60.9999999999]
        printStar(iy);
    }
    system("pause");
    return 0;
}
void printStar(int position) {
    int i;
    for (i=0; i<position-1; i++)
        printf(" ");
    printf("*\n");
}
```





實數平面 寬:2π 高:2	文字模式 寬:80 高:24	繪圖模式視窗 寬:640 高:480
(x,y)	(x1,y1)	(x2,y2)
(0, 0)	(40, 12)	(320, 240)
(π, 0)	(79, 12)	(639, 240)
(-π, 0)	(1, 12)	(1, 240)
(0, 1)	(40, 1)	(320, 1)
(0, -1)	(40, 23)	(320, 479)

實數平面 寬:2π 高:2	文字模式 寬:80 高:24	繪圖模式視窗 寬:640 高:480
x y	x1 = x / π * 39 + 40 y1 = -y * 11 + 12	x2 = x / π * 319 + 320 y2 = -y * 239 + 240

Randomness for Simulation

- Uniform: a discrete random variable in {0, 1}, Pr[0]=Pr[1]=.5

```
#include <time.h>
#include <stdlib.h>
…
srand(time(0));
…
for (i=0; i<100; i++) /* repeat 100 random experiments */
   printf("%d", rand()%2);
```

- Uniform: a continuous random variable in [0, 1) with probability density function f(x) = 1

```
#include <time.h>
#include <stdlib.h>
…
srand(time(0));
…
for (i=0; i<100; i++)
   printf("%f", ((double)rand()) / (RAND_MAX+1.0));
```

Specified Probability Mass/Distribution Function

- Biased coin: Pr[head]=0.3 Pr[tail]=0.7

```
#include <time.h>
#include <stdlib.h>
…
srand(time(0L));
…
for (i=0; i<100; i++) /* head: 1, tail: 0 */
   printf("%d", ((double)rand())/(RAND_MAX+1) >= 0.7 ? 1 : 0);
```

- Cheating dice: {Pr[X=1],…,Pr[X=6]} = {1/4,1/4,1/6,1/6,1/12,1/12}

```
for (i=0; i<100; i++) { /* cdf boundaries: 1/4, 1/2, 2/3, 5/6, 11/12, 1 */
   x = ((double)rand()) / (RAND_MAX+1);
   if (x<0.25) printf("1 ");
   else if (x<0.5) printf("2 ");
   else if (x<2.0/3) printf("3 ");
   else if (x<5.0/6) printf("4 ");
   else if (x<11.0/12) printf("5 ");
   else printf("6 ");
}
```

多重迴圈暴力搜尋

- Search of Perfect Number
- Armstrong Number
- UVa821: Page Hopping