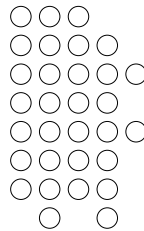


# More Examples w/o Using Arrays

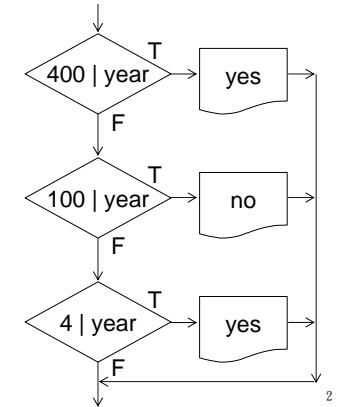
Pei-yih Ting  
NTU math C prog



1

# 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;
}
```



2

# 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;
}
```

3

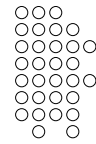
# Check Vowels (cont'd)

## • 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);
}
```

4

## 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;
}
    
```

5

## Check Vowels (cont'd)



- Version 4

```

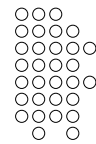
int check_vowel(char ch) {
    char aeiou[] = "aeiouAEIOU";
    int i;
    for (i=0; i<10; i++)
        if (ch == aeiou[i])
            return 1;
    return 0;
}

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

*(aeiou+i)
    
```

6

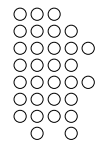
## Test multiples of 11



- 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:  $a_n 10^n + a_{n-1} 10^{n-1} + a_{n-2} 10^{n-2} + \dots \equiv (11 a_n - a_n + a_{n-1}) 10^{n-1} + a_{n-2} 10^{n-2} + \dots \equiv (11(-a_n + a_{n-1}) + a_n - a_{n-1} + a_{n-2}) 10^{n-2} + \dots \equiv a_n - a_{n-1} + a_{n-2} - a_{n-3} + \dots + (-) a_0 \pmod{11}$
- Please write a loop to input the number character by character  
Note: you should use `scanf("%c", ...)` instead of `scanf("%d", ...)`
- You should process each character on the fly w/o using arrays

7

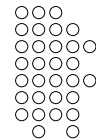
## Approximating $\pi$



- The value for  $\pi$  can be determined by the series equation  
$$\pi = 4 \times (1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + 1/13 - \dots)$$
- Write a program to approximate the value of  $\pi$  using the given formula including terms up through 1/99
- Write a program to approximate the value of  $\pi$  using the given formula until the variation caused by the last term is less than 0.1% of the approximated  $\pi$  value

8

# Calculating $x^n \pmod p$



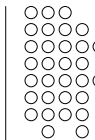
• Let  $x, p$  be two integers and  $n$  be a big integer, calculate  $x^n \pmod p$  (e.g.  $12^{1000000000} \pmod{123457}$ ) efficiently

• If we write  $n = b_k \cdot 2^k + b_{k-1} \cdot 2^{k-1} + \dots + b_0$ ,  
then  $x^n = x^{(((\dots(b_k \cdot 2 + b_{k-1}) \cdot 2 + \dots + b_3) \cdot 2 + b_2) \cdot 2 + b_1) \cdot 2 + b_0}$   
 $= (((\dots((x^{b_k})^2 x^{b_{k-1}})^2 \dots x^{b_3})^2 x^{b_2})^2 x^{b_1})^2 x^{b_0}$

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

• 1st step: derive  $b_0, b_1, \dots, b_k$  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:  $(x^{2^k})^{b_k}$   
 $q = n; xp = x; rs = 1;$   
 while ( $q > 0$ ) {  
     if ( $q \% 2$ )  
          $rs = (rs * xp) \% p;$   
      $q = q / 2;$   
      $xp = (xp * xp) \% p;$   
 }

# Calculate $C_k^n$ and $P_k^n$



• Version 1

```

01 #include <stdio.h> 17 int find_nCk(int n, int k) {
02 #include <stdlib.h> 18 return factorial(n) / (factorial(k)*factorial(n-k));
03 int factorial(int); 19 }
04 int find_nCk(int, int); 20 int find_nPk(int n, int k) {
05 int find_nPk(int, int); 21 return factorial(n)/factorial(n-k);
06 int main(void) { 22 }
07 int n, k, nCk, nPk; 23 int factorial(int n) {
08 printf("Enter the value of n and k\n"); 24 int i, result=1;
09 scanf("%d%d",&n,&k); 25 for (i=1; i<=n; i++)
10 nCk = find_nCk(n, k); 26 result = result*i;
11 nPk = find_nPk(n, k); 27 return result;
12 printf("%dC%d = %d\n", n, k, nCk); 28 }
13 printf("%dP%d = %d\n", n, k, nPk);
14 system("pause");
15 return 0;
16 }
    
```

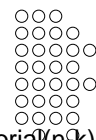
Conceptually correct,  
actually has big problem!!

$$P_3^{15} = 15 \cdot 14 \cdot 13 = 2730$$

$$15! = 1307674368000$$

$$2^{31}-1 = 2147473647$$

# Calculate $C_k^n$ and $P_k^n$



```

• Version 2 long long int
01 #include <stdio.h> 17 int find_nCk(int n, int k) {
02 #include <stdlib.h> 18 return factorial(n)/(factorial(k)*factorial(n-k));
03 long long int factorial(int); 19 }
04 int find_nCk(int, int); 20 int find_nPk(int n, int k) {
05 int find_nPk(int, int); 21 return factorial(n)/factorial(n-k);
06 int main(void) { 22 }
07 int n, k, nCk, nPk; 23 long long int factorial(int n) {
08 printf("Enter the value of n and k\n"); 24 int i; long long int result=1;
09 scanf("%d%d",&n,&k); 25 for (i=1; i<=n; i++)
10 nCk = find_nCk(n, k); 26 result = result*i;
11 nPk = find_nPk(n, k); 27 return result;
12 printf("%dC%d = %d\n", n, k, nCk); 28 }
13 printf("%dP%d = %d\n", n, k, nPk);
14 system("pause");
15 return 0;
16 }
    
```

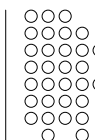
Better! But still quite limited.

$$P_3^{21} = 21 \cdot 20 \cdot 19 = 7980$$

$$21! = 51090942171709440000$$

$$2^{63}-1 = 9223372036854775807$$

# Calculate $C_k^n$ and $P_k^n$



• Version 3

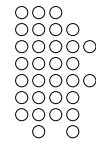
```

P_k^n = n (n-1) ... (n-k+1)
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.  $30P6 = 427518000$   
 $30P7 = 1670497408$   
 $30P8 = -233265280$   
 $100P5 = 444567808$   
 $100P6 = -715731200$

Ex.  $30P13 = 745747076954880000$   
 $30P14 = -5769043765476591616$   
 $100P10 = 7475418734400817152$   
 $100P11 = 8704899442529685504$   
 $100P12 = -27200710659158016$   
 printf("result = %I64d\n", result);

# Calculate $C_k^n$ and $P_k^n$



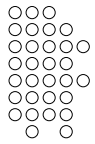
## • Version 3

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

$$C_k^n = \frac{n(n-1) \dots (n-k+1)}{k(k-1) \dots 1}$$

```
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;
}
```

# Calculating $e^x$

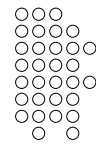


- Approximate  $e^x$  with the formula

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

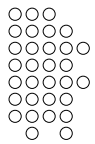
- Do not calculate factorial  $n!$  or  $x^n$  alone again!
- This is a number approximately  $2.71828^x$ , which is larger than  $2^x$ . If  $x$  is large, it will be a huge number. However, when you calculate  $n!$  alone, either it will overflow if you use *int* type; or the representation error is 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 $e^x$



```
01 #include <stdio.h>
02
03 void main()
04 {
05     double x, sum, current;
06     int i;
07     printf("Please input an exponent: ");
08     scanf("%lf", &x);
09     sum = 1 + x; i = 2; current = x * x / i;
10     while (current/sum > 0.0001)
11     {
12         sum += current;
13         i++;
14         current = current * x / i;
15     }
16     printf("e^%lf=%lf\n", x, sum);
17 }
```

# Plot Sine Function in Text Mode



```
01 #include <stdio.h>
02 #include <math.h>
03 #include <stdlib.h>
04
05 void printStar(int position);
06
07 int main(void) {
08     int i;
09     int iy;
10     const double pi = atan(1.0)*4;
11     for (i=0; i<30; i++) {
12         iy = (sin(i/30.0*2*pi)+1) * (30-1e-10) + 1; // [1.0, 60.999999999]
13         printStar(iy);
14     }
15     system("pause");
16     return 0;
17 }
```

```
18 void printStar(int position) {
19     int i;
20     for (i=0; i<position-1; i++)
21         printf(" ");
22     printf("*\n");
23 }
```

# Randomness for Simulation



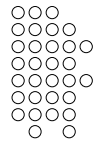
- Uniform: a discrete random variable in  $\{0, 1\}$ ,  $\Pr[0]=\Pr[1]=.5$

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

# Specified Probability Mass/Distribution Function



- Biased coin:  $\Pr[\text{head}]=0.3$   $\Pr[\text{tail}]=0.7$

```
#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 >= 0.7 ? 1 : 0);
```

- Cheating dice:  $\{\Pr[X=1], \dots, \Pr[X=6]\} = \{1/4, 1/4, 1/6, 1/6, 1/12, 1/12\}$   
for (i=0; i<100; i++) { /\* cdf boundaries: 1/4, 1/2, 2/3, 5/6, 11/12, 1 \*/  
 x = ((double)rand()) / RAND\_MAX;  
 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 ");  
}