# Lập chương trình cho máy tính

**Ngôn ngữ lập trình C - Giới thiệu**

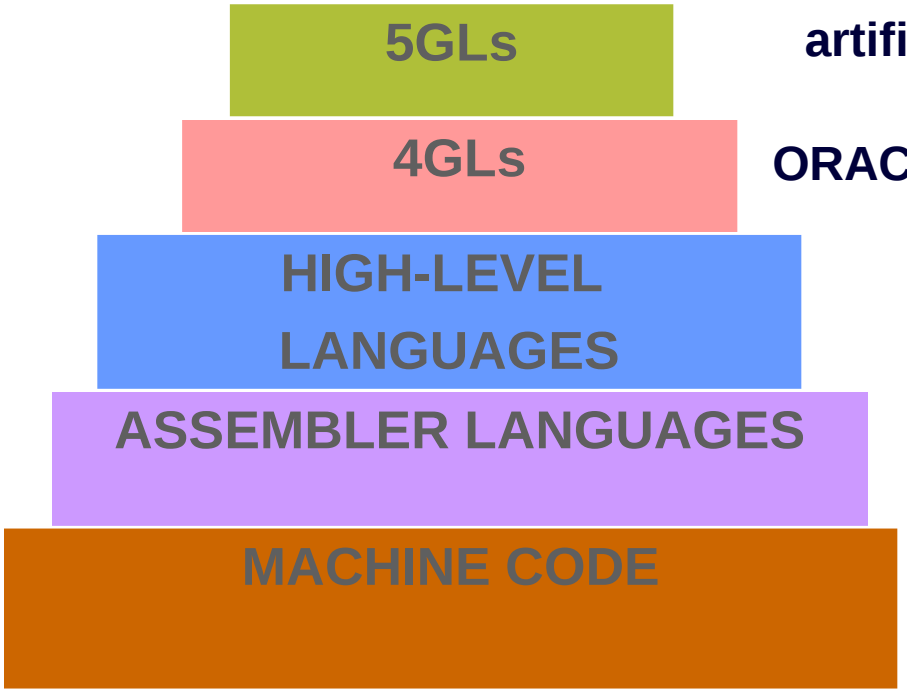
# Tài liệu tham khảo


- Bài giảng: Kỹ thuật lập trình. Lưu Nguyễn Kỳ Thư, Tân Hạnh. Khoa CNTT2, Học viện CNBCVT.
- Ngôn Ngữ Lập Trình C. Quách Tuấn Ngọc. Nhà Xuất Bản Giáo Dục, 1998.
- Efficient C programming. Mark Allen Weiss. Prentice Hall, 1998.
- Introduction to Computing System, from Bits and Gates to C and Beoynd. Yale N. Patt, Sanjay J. Patel. McGrawHill, 1999.

# Một số khái niệm

- Computer program –chương trình máy tính là một tập các câu lệnh (instruction) hướng dẫn máy tính làm một số việc nhất định.
- Programming language - Ngôn ngữ lập trình là ngôn ngữ để viết chương trình. Có nhiều loại ngôn ngữ lập trình.
- Compiler – trình biên dịch, là phần mềm chịu trách nhiệm dịch chương trình viết bằng một ngôn ngữ lập trình sang dạng mã máy.

# Các lớp Ngôn ngữ lập trình




**artificial intelligence**

**ORACLE, SEQUEL, INGRES, ...**

**ForTran, COBOL, C, C++,**
**LISP, Pascal, Java, ...**

# Thuật toán - Algorithm

- Tập các lệnh được tổ chức có thứ tự  nhằm giải quyết một bài toán hoặc đạt đến một mục tiêu nào đó.
- Ví dụ:
  - hướng dẫn chế biến một món ăn,
  - hướng dẫn sửa chữa xe máy,
  - cách giải một bài toán.
  - …
- Algorithm –Thuật toán - Thuật giải

# Thuật giải tốt

- Một thuật giải tốt là thuật giải:
  - chính xác
  - rõ ràng
  - đúng
  - hiệu quả
  - và có thể bảo trì được.

- Chúng ta có thể viết một thuật giải cho máy tính bằng ngôn ngữ bình thường nhưng có thể không rõ ràng. Thay vào đó, chúng ta sẽ dùng ngôn ngữ lập trình (hoặc một ngôn ngữ giả lập ngôn ngữ lập trình gọi là mã giả pseudocode)

# Tính điểm trung bình môn học

- Nhập: điểm thực hành Vật Lý, điểm bài tập, điểm bài kiểm tra giữa học kỳ, điểm bài kiểm tra cuối học kỳ.

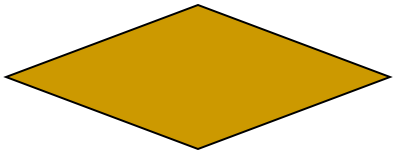
| | | Điểm | | hệ số |
|---|---|---|---|---|
| Thực hành : | | 8 | | 2 |
| bài tập: | | 9 | | 2 |
| KT giữa kỳ: | 8 | | 4 | |
| KT cuối kỳ: | | 8 | | 6 |

- Tổng cộng: TONG = 8*2 + 9*2 + 8*4 + 8*6
- Điểm trung bình:  TB = TONG/(2+2+4+6)

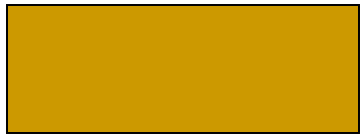
# Sơ đồ xử lý

- Sử dụng sơ đồ xử lý để minh họa quá trình xử lý một chương trình.

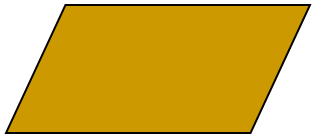
**start,stop**



**condition expression**



**process**



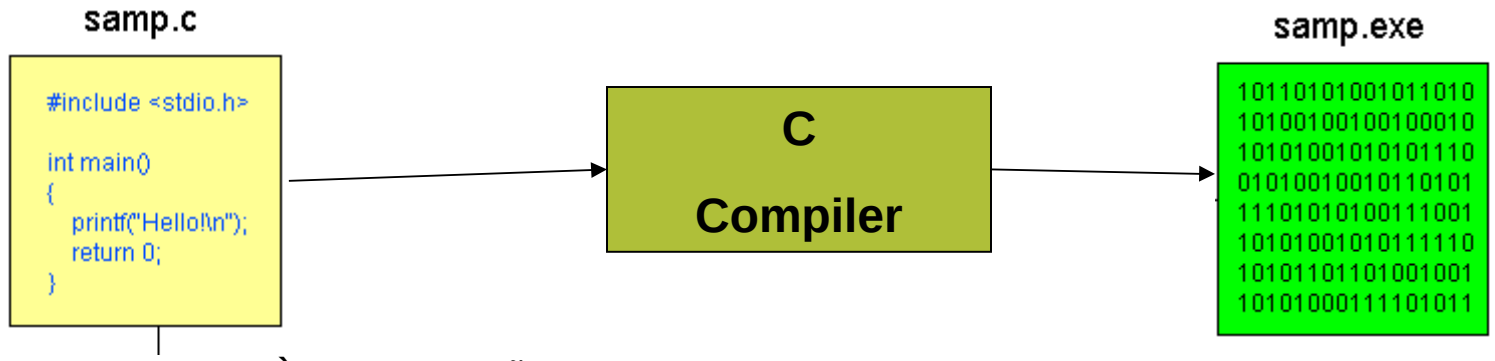
**data**

**flow**

Bài tập: dùng sơ đồ để biểu diễn bài toán nhập và tính điểm trung bình.

# Ngôn ngữ lập trình C

- có thể đọc và viết mã chương trình trên hầu hết các hệ thống.
- chuyển lên C++ và có thể viết các kịch bản CGI (CGI script) cho các Website.
- C là ngôn ngữ biên dịch (complied language).




**Viết chương trình bằng ngôn ngữ C bằng các chương trình soạn thảo (Notepad, copy con, các công cụ viết chương trình)**

**Không dùng các chương trình soạn thảo văn bản (vd:Word, WordPad)**

# Lập chương trình cho máy tính

## Ngôn ngữ lập trình C – Khái niệm cơ sở

**Biến, Hằng, Toán tử, Kiểu dữ liệu cơ sở, Các phép toán và Các từ khóa**

Lê Hà Thanh

Học kỳ 2, 2004-2005

# Chương trình C đầu tiên

```
1. #include <stdio.h>
2.
3. int main()
4. {
5.     printf(“Hello\n");
6.     return 0;
7. }
```

# Chương trình C

- #include <stdio>
  - khai báo sử dụng thư viện xuất/nhập chuẩn (standard I/O library). Các thư viện khác: string, time, math…
- int main()
  - khai báo hàm main(). Chương trình C phải khai báo (duy nhất) một hàm main(). Khi chạy, chương trình sẽ bắt đầu thực thi ở câu lệnh đầu tiên trong hàm main().
- { … }
  - mở và đóng một khối mã.
- printf
  - hàm printf() gửi kết xuất ra thiết bị xuất chuẩn (màn hình). Phần nằm giữa "…" gọi là chuỗi định dạng kết xuất (format string)
- return 0;
  - ngừng chương trình. Mã lỗi 0 (error code 0) – không có lỗi khi chạy chương trình.

# Mở rộng 1

```
1.  #include <stdio.h>
2.
3.  int main()
4.  {
5.      int a, b, c;
6.      a = 5;
7.      b = 7;
8.      c = a + b;
9.      printf("%d + %d = %d\n", a, b, c);
10.     return 0;
11. }
```

# Biến (variable)

- dùng để giữ các giá trị.
- Khai báo: <type> <var-name>;

  vd: int b;

- Gán giá trị vào biến:
  <var-name> = <value>;

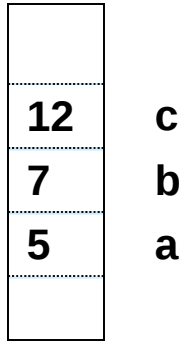
  vd: b = 5;

- Sử dụng biến:

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

# Mở rộng 2

```
1.  #include <stdio.h>
2.
3.  int main()
4.  {
5.      int a, b, c;
6.      printf("Nhap so thu nhat: ");
7.      scanf("%d", &a);
8.      printf("Nhap so thu hai: ");
9.      scanf("%d", &b);
10.     c = a + b;
11.     printf("%d + %d = %d\n", a, b, c);
12.     return 0;
13. }
```




```
C:\> tong.exe

    Nhap so thu nhat:    5

    Nhap so thu hai:     7

    5 + 7 = 12

C:\>_
```

# Chú ý

- C phân biệt chữ hoa/chữ thường do đó phải viết đúng tên lệnh.
  vd: printf chứ không phải là Printf, pRintf, PRINTF.

- Trong câu lệnh scanf() để lấy giá trị vào biến, phải luôn dùng dấu & trước tên biến.

- Khi gọi các hàm phải khai báo các tham số đúng vị trí và đầy đủ.

- Phải khai báo biến trước khi sử dụng trong chương trình.

# Các Toán tử

| Priority | Category | Example | Associativity |
|---|---|---|---|
| 0 | Primary expression | *identifiers constants* | None |
| 1 | Postfix | Function() () [] -> | left to right |
| 2 | Prefix and unary | ! ~ + - ++ -- & sizeof | right to left |
| 2 | Type cast | ( typeName ) | right to left |
| 3 | Multiplicative | * / % | left to right |
| 4 | Additive | + - | left to right |
| 5 | Shift | << >> | left to right |
| 6 | Relational | < <= > >= | left to right |
| 7 | Equality | == != | left to right |
| 8 | Boolean AND | & | left to right |
| 9 | Boolean XOR | ^ | left to right |
| 10 | Boolean OR | \| | left to right |
| 11 | Logical AND | && | left to right |
| 12 | Logical OR | \|\| | left to right |
| 13 | Conditional operator | ? | Right to left |
| 14 | Assignment | = *= /= %= += -= &= \|= ^= <<= >>= | right to left |

# Các toán tử so sánh và toán tử logic

| | | Relational and Quality Operators | | | | | | Possible Mistakes | | |
|---|---|---|---|---|---|---|---|---|---|---|
| X | Y | *X < Y* | *X <= Y* | *X > Y* | *X >= Y* | *X != Y* | *X == Y* | X=Y | X<<Y | X>>Y |
| 3 | 3 | 0 | 1 | 0 | 1 | 0 | 1 | 3 | 24 | 0 |
| 3 | 4 | 1 | 1 | 0 | 0 | 1 | 0 | 4 | 48 | 0 |
| 4 | 3 | 0 | 0 | 1 | 1 | 1 | 0 | 3 | 32 | 0 |

| | | Logical Operators | | | | Possible Mistakes | |
|---|---|---|---|---|---|---|---|
| X | Y | X && Y | X \|\| Y | !X | !Y | X & Y | X \| Y |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 7 | 0 | 1 | 1 | 0 | 0 | 7 |
| 5 | 0 | 0 | 1 | 0 | 1 | 0 | 5 |
| 5 | 7 | 1 | 1 | 0 | 1 | 5 | 7 |
| 8 | 7 | 1 | 1 | 0 | 1 | 0 | 15 |

# Các kiểu dữ liệu cơ bản

- Integer: int (các giá trị nguyên 4-byte)
- Floating point: float (các giá trị dấu chấm động 4-byte)
- Character: char (ký tự 1-byte)
- Double: double (dấu chấm động 8-byte)
- Short: short (số nguyên 2-byte)
- unsigned short (số nguyên không dấu)
- unsigned int

# Biến và hằng số

- Biến số (variable) được dùng để giữ các giá trị và có thể thay đổi các giá trị mà biến đang giữ

- Khai báo: <typename> varname;
  Vd:

```
int i;
float x, y, z;
char c;
```

- Gán giá trị cho biến: <varname> = <value>;
  vd:

```
i = 4;
x = 5.4;
y = z = 1.2;
```

# Hằng số

- Hằng số (constant) giá trị không thay đổi trong quá trình sử dụng.

- Khai báo hằng:
  #define <constantname> <value>

  vd:
  #define TRUE 1
  #define FALSE  0

# Kiểu và chuyển kiểu (typecasting)

- C cho phép chuyển đổi kiểu dữ liệu cơ bản trong khi đang tính toán.
- ví dụ:

```
void main()
{
    float a;
    int b;
    b = 10/3;
    a = (float)10/3;
    printf("a = %f \n b = %d\n", a, b);
}
```

- Chú ý: khi thực hiện chuyển kiểu có thể gây ra mất ý nghĩa dữ liệu

# Định nghĩa kiểu (typedef)

- Có thể định nghĩa các kiểu riêng bằng lệnh typedef.

vd:

```
#define TRUE 1
#define FALSE  0
typedef int boolean;

void main() {
boolean b;
b = FALSE;
/*...*/
}
```

# Các phép toán số học

- + - / *
- %: phép chia lấy phần dư trong số nguyên. (modulo).
- i = i + 1; i++; ++i;
- i = i – 1; i--; --i;
- i = i + 3; i += 3;
- i = i * j; i *= j;

# Mở rộng 1

```
1.  #include <stdio.h>
2.
3.  int main()
4.  {
5.      int a, b, c;
6.      a = 5;
7.      b = 7;
8.      c = a + b;
9.      printf("%d + %d = %d\n", a, b, c);
10.     return 0;
11. }
```

# Lập chương trình cho máy tính

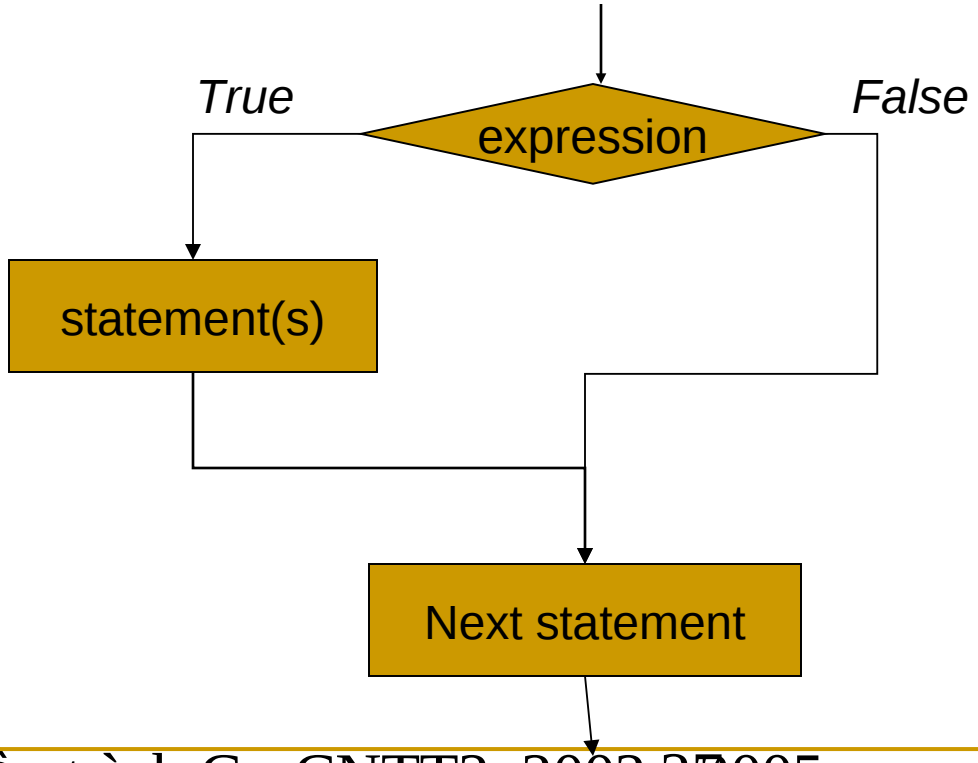
**Các cấu trúc điều khiển**

Lê Hà Thanh

Học kỳ 2, 2004-2005

# Câu lệnh điều kiện if

```
if (<dieu kien>)
    {
    /* cac lenh thuc hien neu dieu kien dung */
    }
…
```

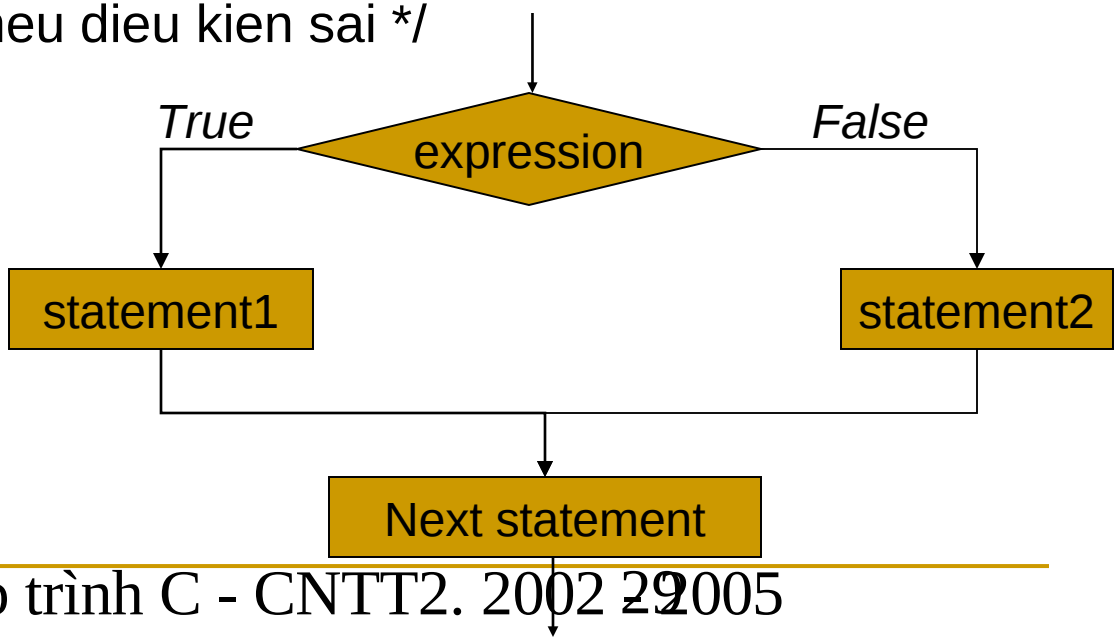


# Ví dụ

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

2. int main() {
3.    int b;

4.    printf("Enter a value:");
5.    scanf("%d", &b);
6.    if (b < 0)
7.             printf("The value \
                              is negative\n");
8.    return 0;
9. }
```

# if … else …

```
if (<dieu kien>)
        {
        /* cac lenh thuc hien neu dieu kien dung */
        }
else
        {
        /* cac lenh thuc hien neu dieu kien sai */
        }
    …
```



# Ví dụ

```
…
printf(“1/X is: “);
if(X)
    printf(“ %f \n”, 1/X);
else
    printf(“ undefined        \n”);
…
```

# Lỗi đơn giản nhưng dễ phạm

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

2.  int main() {
3.      int b;

4.      printf("Enter a value:");
5.      scanf("%d", &b);
6.      if (b == 5)
7.          printf(“b is ");  printf( “5 \n”);
8.      return 0;
9.  }
```

# Lỗi đơn giản nhưng dễ phạm

1. printf(“1/X is: “);
2. if(X < 0) ;
3. printf(“ X is negative \n”);
4. …

# Ví dụ: Kiểm tra nhiều điều kiện

```
1.  #include <stdio.h>
2.  int main() {
        int b;

3.      printf("Enter a value:");
4.      scanf("%d", &b);
5.      if (b < 0)
6.              printf("The value is negative\n");
7.      else if (b == 0)
8.              printf("The value is zero\n");
9.      else
10.             printf("The value is positive\n");
11.     return 0;
12. }
```

- Bài tập: Viết chương trình giải phương trình bậc nhất:
  ax + b = 0. Biện luận các điều kiện có nghiệm của phương trình.

# Điều kiện lồng nhau

- Câu lệnh if có thể được lồng vào nhau.

```
1. if ( X >= 0 ) {
2.     if ( Y < 0 )
3.                 Y = Y + sqrt(X);
4. }
5. else
6.     Y = Y + sqrt(-X);
```

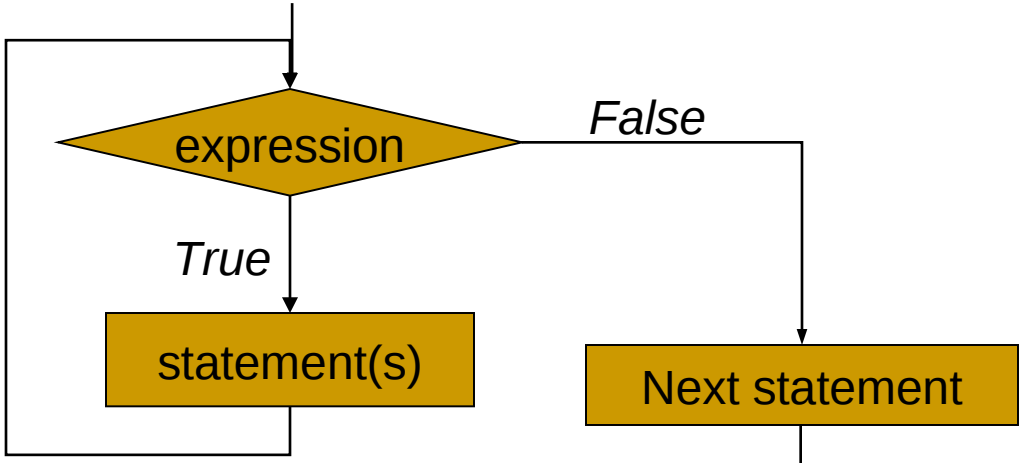
- Tuy nhiên, cần chú ý đến thứ tự các cặp lệnh if … else … khi lồng các lệnh if. Nếu không sẽ phát sinh lỗi.

```
1. if ( X >= 0 )
2.     if ( Y < 0 )
3.                 Y = Y + sqrt(X);
4. else
5. Y = Y + sqrt(-X);
```

- Bài tập: Viết chương trình giải phương trình bậc 2: ax^2 + bx +c = 0. Chú ý các điều kiện có nghiệm.

# Lặp - lệnh while

- while (bieu thuc dieu kien)
  {cac lenh}
  - Khi biểu thức điều kiện (expression) còn khác 0 (TRUE), lệnh (statement) tiếp tục được thực hiện. Nếu expression bằng 0 (FALSE), lệnh while dừng và chương trình sẽ gọi lệnh kế tiếp sau while.
  - Nếu lúc đầu expression bằng 0 thì (statement) trong while không bao giờ được gọi thực hiện.



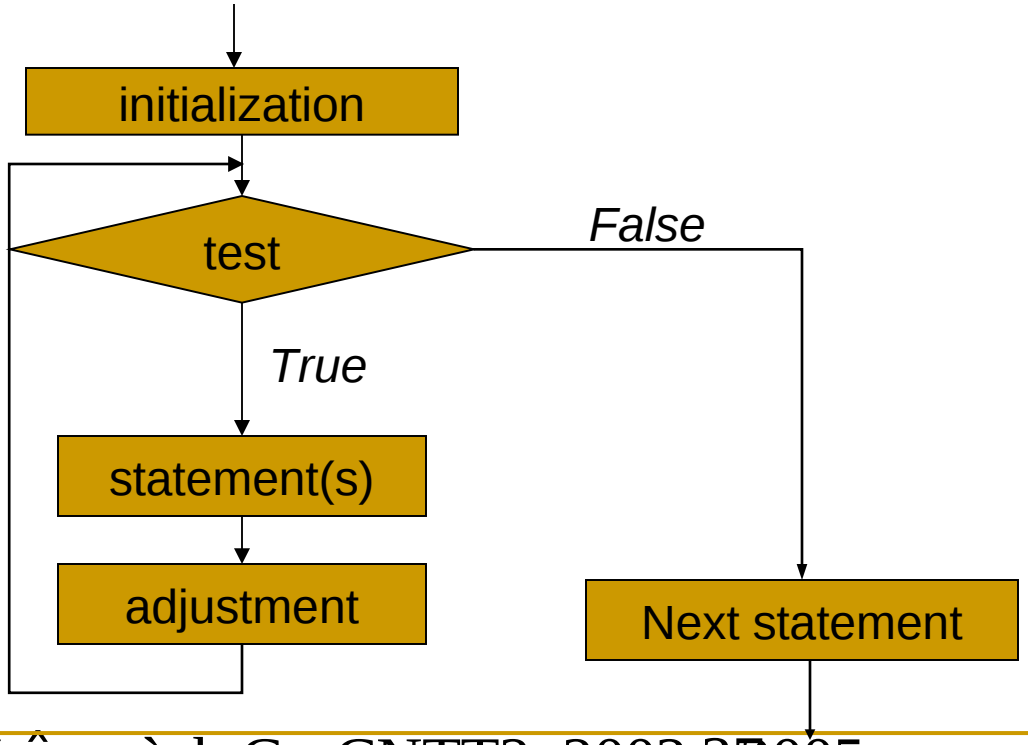
# Ví dụ

- In bảng đổi nhiệt độ từ độ Fahrenheit (oF) sang độ Celcius (oC).

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

2. int main() {
3. int a = 0;
4. while (a <= 100) {
5.     printf("%4d degrees F = %4d degrees C\n",a, (a - 32)*5/9);
6.     a = a + 10;
7.     }
8. return 0;
9. }
```

# Lặp - lệnh for

- for (initialization; test; adjustment) {statement(s)}
  - Khởi động. Sau đó, nếu điều kiện (test) khác 0: lệnh (statement) được thi hành, lệnh điều chỉnh lại "biến đếm" được gọi thi hành.



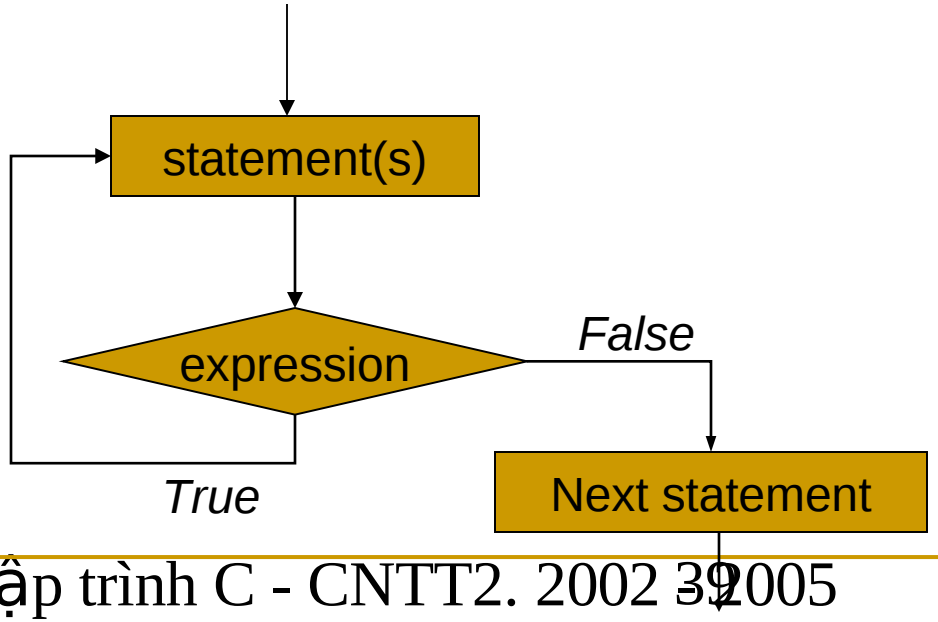
# Ví dụ

- Bài toán đổi nhiệt độ. Yêu cầu: hiển thị nhiệt độ chính xác đến con số thập phân sau dấu phẩy.

```
1.  #include <stdio.h>
2.  int main() {
3.  float a = 0;
4.  int i;
5.  for(i=0; i<=100; i+=10) {
6.          printf("%6.2f degrees F = %6.2f degrees C\n",
                                                  a, (a - 32.0) * 5.0 / 9.0);
7.          a = a + 10;
8.          }
9.  return 0;
10. }
```

# Lặp - lệnh do  while

- do
  {statement(s)}
  while (expression) ;
  - Thực hiện lệnh (statement). Kiểm tra biểu thức điều kiện (expression). Nếu (expression) bằng 0, dừng. Nếu không, thực hiện (statement).
  - Lệnh do while thực hiện (statement) ít nhất một lần.



# Ví dụ - giao diện chương trình

```
1.  #include <conio.h>
2.  #include <stdio.h>
3.  #define PTB1        1
4.  #define PTB2        2
5.  #define STOP        3

6.  int main()
7.  {
8.      int i;
9.      do {
10.         clrscr();       // xoa man hinh
11.         printf("  Chuong trinh giai phuong trinh bac thap \n");
12.         printf("  1. Giai phuong trinh bac 1: ax + b = 0 \n");
13.         printf("  2. Giai phuong trinh bac 2 : ax^2 + bx + c = 0 \n");
```

```
14.         printf("  3. Thoat chuong trinh \n\n");
15.         printf("  Chon muc so (1/2/3) ? ");
16.         scanf("%d", &i);
17.         if(i == PTB1)
18.             printf("Giai phuong trinh bac 1: hien chua co\n");
19.         else if(i == PTB2)
20.             printf("Giai phuong trinh bac 2: chua cai dat\n\n");
21. } while (i != STOP);

22. return 0;
23. }
```

- Bài tập: Ghép chương trình trên với hai chương trình trong bài tập 1 và 2

# break

- dùng để thoát khỏi vòng lặp giữa chừng.

  cú pháp:
  break;

- Thường sử dụng cùng với lệnh if để kiểm tra điều kiện dừng trước khi dùng lệnh break.

- Bài tập: Viết chương trình nhập vào một số rồi tìm số nguyên tố đầu tiên lớn hơn số vừa nhập

# Tìm số nguyên tố lớn

```
1.  #include <stdio.h>
2.  #define TRUE 1
3.  main(void)
4.  {
5.      unsigned long int Divisor, PossiblePrime;
6.      int FoundPrime;

7.      printf("Enter the starting number: ");
8.      scanf("%lu", &PossiblePrime);
9.          if(PossiblePrime <= 2)
10.             PossiblePrime = 2;
11.         else
12.             if(PossiblePrime !=3 )
13.                 {
14.                     if(PossiblePrime %2 == 0)
15.                         PossiblePrime++; /* Need an odd number */
```

```
16.                 for( ; ; PossiblePrime += 2)
17.                     {
18.                     FoundPrime = !TRUE;
19.                     for(Divisor = 3;PossiblePrime % Divisor;Divisor += 2)
20.                     if(Divisor * Divisor > PossiblePrime)
21.                         {   FoundPrime = TRUE;
22.                             break;
23.                         }
24.                     if (FoundPrime)
25.                         break;
26.                 }
27.             }
28.         printf("Next largest prime is %lu\n", PossiblePrime);
29. }
```

# continue

- bỏ qua các lệnh kế tiếp trong một vòng lặp và bắt đầu vòng lặp tiếp theo.

  cú pháp:
  ```
  continue;
  ```

- chỉ áp dụng với lệnh lặp.

- Bài tập: Viết chương trình nhập vào một số và tìm ra tất cả các thừa số nguyên tố của số đó.

# Tìm thừa số nguyên tố

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

2.  main(void)
3.  {
4.      unsigned long NumberToFactor, PossibleFactor, UnfactoredPart;

5.      printf("Enter the number to factor: ");
6.      scanf("%lu", &NumberToFactor);

7.      PossibleFactor = 2;
8.      UnfactoredPart = NumberToFactor;

9.      while(PossibleFactor * PossibleFactor <= UnfactoredPart)
10.         {
11.         if(UnfactoredPart % PossibleFactor == 0)
```

```
12.             { /* Found a factor */
13.             printf("%lu", PossibleFactor);
14.             UnfactoredPart /= PossibleFactor;
15.             continue;
16.         }
17.     /* No factor: try next factor */
18.     if(PossibleFactor == 2)
19.         PossibleFactor = 3;
20.     else
21.         PossibleFactor += 2;
22.     }
23.     /* print last factor */
24.     printf("%lu\n", UnfactoredPart);
25. }
```

# Lệnh switch

- Bài tập:
  Viết chương trình lấy ngẫu nhiên 1000 số nguyên và đếm số lần xuất hiện ở hàng đơn vị các số chẵn (2, 4, 6, 8), số lẻ (1, 3, 5, 7, 9) và số 0.

- Nếu chúng ta dùng cấu trúc lệnh if ... else ... if … thì phức tạp và có thể đòi hỏi nhiều phép thử.
  Lý do: if ... else ... : rẽ nhánh hai chiều.

- Thử cài đặt bài toán bằng if...else...

# Lệnh switch

- Dùng lệnh switch để cài đặt cơ chế rẽ nhánh nhiều chiều.
  cú pháp:

```
switch(<expression>)
        {
                case case1:
                case case2:
                        <statements>;
                        break;
                /* … */
                case casen:
                        <statements>;
                        break;
                default:
                        <statements>;
                        break;
        }
```

# Giải bài bằng switch

```
1.  #include <stdlib.h>
2.  #include <stdio.h>
3.  #include <time.h>

4.  int main(void)
5.  {   int n;
6.      int n_even = n_odd = n_zero = 0;

7.      randomize();
8.      for(int i=0; i<1000; i++)
9.      {   n = random(1000);
10.         switch (n%10) {
11.             case 2:
12.             case 4:
13.             case 6:
14.             case 8:
15.                     n_even++;
16.                     break;
```

```
17.             case 1:
18.             case 3:
19.             case 5:
20.             case 7:
21.                 n_odd++;
22.                 break;
23.             case 0:
24.                 n_zero++;
25.                 break;
26.         }
27.     }
28.     // print out the summary
29.     printf("Number of even_eding number: %d\n"\
                    Number of odd_ending number: %d\n"\
                    Number of zero_ending number: %d\n",
                    n_even, n_odd, n_zero);
30.     return 0;
31. }
```

# Một số toán tử và lệnh khác

- Toán tử ‘,’ được dùng để khởi động nhiều biến trong vòng lặp.
  Ví dụ:

```
for(i = 0, j = 0; i < 5; i++, j += i++)
        printf("i = %d,  j = %d, i+j = %d\n", i, j, i+j);
```

- Kết quả là: ????

- Toán tử ba ngôi

```
<TestExpr> ? <YesExpr> : <NoExpr>
```

  Ví dụ:

```
Max = (Y > Z) ? Y : Z;
```

# Một số toán tử và lệnh khác

- Lệnh goto cho phép nhảy không điều kiện đến bất kỳ nơi nào trong chương trình.
  Cú pháp:

  goto <label>

  Ví dụ: xem chương trình ví dụ.

- Lệnh goto làm mất cấu trúc chương trình.

# Từ khóa của C

| | | | |
|---|---|---|---|
| auto | break | case | char |
| const | continue | default | do |
| double | else | enum | extern |
| float | for | goto | if |
| int | long | register | return |
| short | signed | sizeof | static |
| struct | switch | typedef | union |
| unsigned | void | volatile | while |

# Từ khóa của C

- #define để khai báo hằng số và …
- typedef để khai báo kiểu dữ liệu riêng

- Toán tử sizeof xác định số byte được dùng để chứa một đối tượng

  ví dụ:

```
typedef unsigned long int Int32;
/* ... */
int x;
x = sizeof(Int32);  // x = 4
```

# Lập chương trình cho máy tính

**Hàm (function)**

Lê Hà Thanh
Học kỳ 2, 2004-2005

# Hàm (function)

- Sản xuất bằng cách lắp ghép các module: các module được lắp ghép lại thành sản phẩm, các module có thể được cải tiến nhưng không ảnh hưởng đến các module khác trong sản phẩm.

*Với chương trình máy tính*

- Phân chia chương trình thành các phần nhỏ - các chương trình con (routine) hay còn gọi là các hàm (function)
- Cách tiếp cận phân tích bài toán theo hướng top-down: xác định chức năng của các hàm.
- Các hàm có thể được dùng lại nhiều lần → thành lập các thư viện hàm. (vd: stdio, stdlib, conio, math, string,…)
- Một chương trình C là một tập hợp các hàm tương tác bằng cách gọi lẫn nhau và truyền các thông tin qua lại giữa các hàm.
- Với chương trình đơn giản, tất cả các xử lý nên được đặt trong hàm main.

# Các thành phần của hàm

- Tên hàm (name)
- danh sách tham số (list of parameters)
- kiểu trả về (return type)
- thân hàm (function body)
- lệnh trả về (return)

**giao diện (*interface*) của hàm**

  - <return_type> function_name (<list_of_parameters>)

- Các hàm phải được khai báo trước khi được gọi thi hành.

# Thành phần của hàm – Tên hàm

- Tên hàm là một định danh (identifier), do đó nó tuân theo các quy định của ngôn ngữ C cho định danh. (xem bảng các toán tử)
- Nên đặt tên có ý nghĩa.
- Không đặt tên trùng với tên các hàm hệ thống trong C hoặc các từ khóa của C.

# Danh sách tham số

- Danh sách tham số xác định các đối số được đưa vào hàm.
- Các đối số được khai báo trong phần mô tả cài đặt của hàm thì được gọi là các tham số hình thức (formal parameters).
- Mỗi tham số hình thức là một cặp: <type> <identifier>. Từ khoá void có thể được dùng nếu không có tham số hình thức nào cần khai báo. Các tham số trong các hàm khác nhau có thể trùng tên.
- Khi gọi hàm, các đối số đưa vào hàm phải đầy đủ và đúng kiểu như đã khai báo.

# Ví dụ

```
/* ... */
1. float max(float x, float y)
2. {
3.         return (x > y ? x : y);
4. }
/*…*/

6. int main()
7. {
8.         float z = 4.7;
9.         float x = max(4.5, z);
10. }
```

# Giá trị trả về (return value)

- Một hàm được phép trả về cho phần chương trình gọi nó một giá trị: giá trị trả về.
- Chương trình gọi hàm có thể sử dụng giá trị trả về.
- Một số hàm không cần trả về các giá trị. Từ khóa void được dùng trong khai báo giá trị trả về của các hàm này.
- Kiểu int sẽ là kiểu của trị trả về nếu không chỉ rõ kiểu giá trị trả về trong khai báo hàm.

ví dụ:

```
afunction() { /*…*/ }
```

# Thân hàm (function body)

- { /* các đoạn mã trong thân hàm */ }
- Các biến có thể được khai báo bên trong hàm → biến cục bộ (local variable)
- Biến cục bộ không được trùng tên với tham số hình thức trong khai báo hàm.
- Các biến cục bộ chỉ có giá trị trong phạm vi của hàm.

# Phạm vi truy cập của biến

- Phạm vi truy cập (scope) của biến xác định vùng chương trình có thể truy cập đến biến.
- Biến được khai báo trong khối lệnh (nằm giữa { }) có thể được truy cập bởi các lệnh nằm trong cùng khối và các lệnh thuộc các khối con.
- Biến được khai báo "ngoài cùng" có phạm vi truy cập trong toàn chương trình → biến toàn cục (global variables).
- Biến cục bộ (local variable) được khai báo và sử dụng trong phạm vi một khối lệnh và các khối lệnh con.
- Biến thuộc phạm vi trong cùng được tham chiếu đến đầu tiên.

# Ví dụ

```
1.  int i=1;        /* i là biến toàn cục vì nằm ở ngoài các khối lệnh */

2.  { /* block A */
3.        int i=2;
4.        printf (“%d\n”, i); /* outputs 2 */ }
5.  { /* Block B */
6.        int i=3;
7.        printf (“%d\n”, i); /* outputs 3 */

8.        { /* Block C */
9.              int i=4;
10.             printf (“%d\n”, i); /* outputs 4 */ }
11.       { /* Block D */
12.             printf (“%d\n”, i); /* outputs 3 */ }
13. }
14. { /* Block E */
15.       printf (“%d\n”, i); /* outputs 1 */ }
```

# Lệnh return

- kết thúc hàm và trả quyền điều khiển về cho phần chương trình có lời gọi hàm.

- cú pháp:

  return Expr;

  hoặc

  return;

- hàm tự kết thúc khi thực hiện hết lệnh cuối cùng.

# Truyền tham số khi gọi hàm

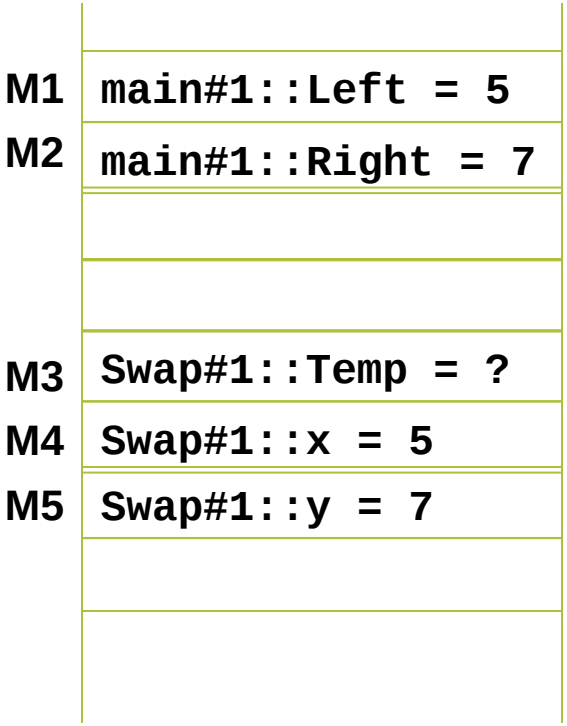
- Truyền tham chiếu (*call by reference*): các tham chiếu đến các tham số hình thức là tham chiếu đến các đối số. Giá trị của các đối số có thể được thay đổi từ xử lý bên trong hàm.

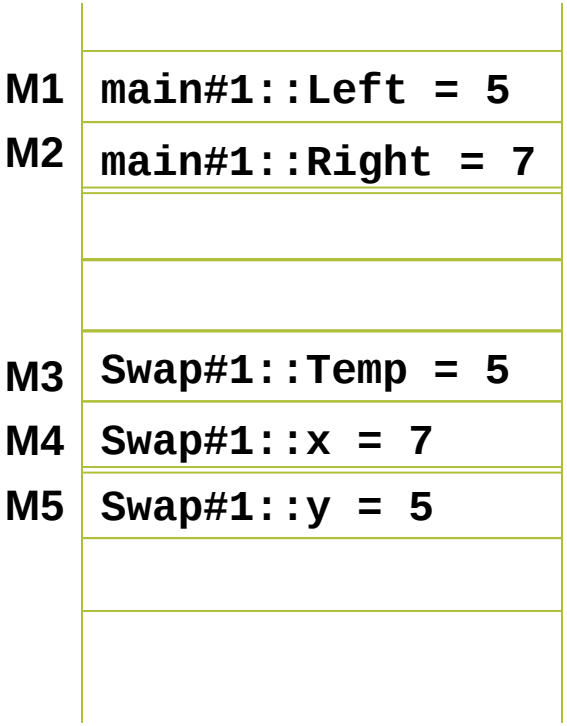
- Truyền giá trị (*call by value*): các đối số đưa vào hàm được chép vào các tham số hình thức. Các giá trị của các đối số được sử dụng trong hàm nhưng những thay đổi của tham số hình thức trong hàm không làm thay đổi giá trị của các đối số truyền vào.

# Truyền giá trị

```
1.  /* Swapping routine that doesn't work */
2.  #include <stdio.h>

3.  void Swap(int x, int y)
    {   int Temp;

4.      Temp = x;
5.      x   = y;
6.      y   = Temp;
7.  }

8.  main(void)
9.  {   int Left, Right;

10.     Left = 5; Right = 7;
11.     Swap(Left, Right);
12.     printf("Left = %d, Right = %d\n", Left, Right);
13. }
```

# Tại sao truyền giá trị không làm thay đổi giá trị đối số



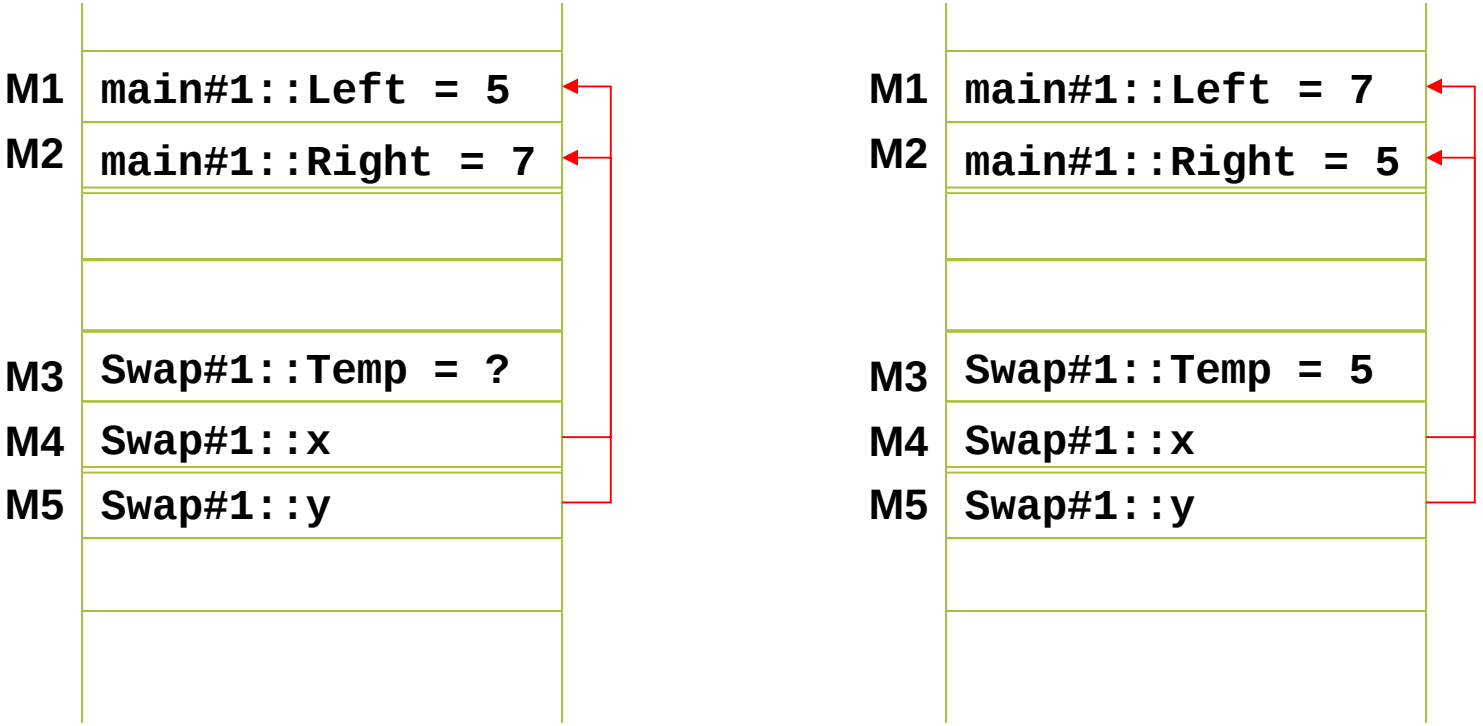



# Truyền bằng tham chiếu

- /* Swapping routine that does work */
- #include <stdio.h>
- void Swap(int &x, int &y)

```
{
        int Temp;

        Temp    = x;
        x       = y;
        y       = Temp;
}

main(void)
{
        int Left, Right;

        Left = 5; Right = 7;
        Swap(Left, Right);
        printf("Left = %d, Right = %d\n", Left, Right);
}
```

# Tại sao truyền bằng tham chiếu làm thay đổi giá trị đối số



# Khai báo hàm trước (Prototyping)

- Function prototype: khai báo trước dạng hàm (kiểu trả về, tên hàm, danh sách tham số) sẽ được gọi trong đoạn mã.

- Phép kiểm tra kiểu sẽ không phát sinh lỗi nếu các hàm được khai báo trước.

- Ví dụ:
  trong ví dụ về khai báo hàm

```
int Max(int x, int y);
int Min(int x, int y);
```

- Sử dụng hàm như tham số.

- Ví dụ:

  bài tập viết chương trình giải phương trình bậc hai.

```
…
x1 = (-b + sqrt(delta))/(2*a);
…
```

# Dừng chương trình và mã lỗi

- Thông thường main trả về giá trị kiểu int
- có thể sử dụng khai báo: void main()
- Nên sử dụng giá trị trả về để kiểm soát xử lý của chương trình.
- Sử dụng hàm exit(<exitcode>); để dừng chương trình và trả về mã lỗi.
- Nên xây dựng một đoạn chương trình con làm nhiệm vụ bắt lỗi trong quá trình chạy.

# Lập chương trình cho máy tính

**Đệ Quy**

Lê Hà Thanh
Học kỳ 2, 2004-2005

# Đệ quy

- Ví dụ: Viết chương trình nhập số tự nhiên n và tính giai thừa : n!.
- Giải quyết bài toán bằng vòng lặp

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

2.  unsigned long int factorial(int n)
3.  {   unsigned long f = 1;
4.      for (int i = 1; i<=n; i++)
5.          f *= i;
6.      return f;
7.  }

8.  int main(void)
9.  {   int n;

10.     printf("Nhap n:"); scanf("%d", &n);
        printf("n! = %d! = %l\n", n, factorial(n));
11.     return 0;
12. }
```

# Đệ quy

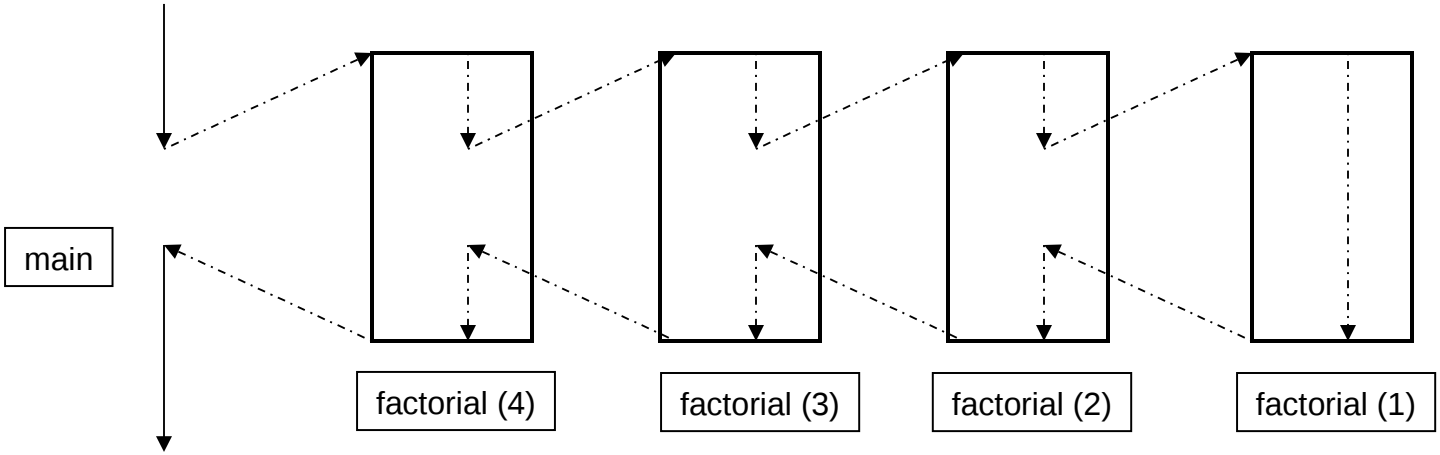
- Một hàm được gọi là đệ quy nếu như trong quá trình xử lý, hàm này có một lời gọi đến chính nó.
- Giải quyết bài toán bằng đệ quy

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

2.  unsigned long int factorial(int n)
3.  {   if(n==0)
4.          return 1;
5.      return (n* factorial(n-1));
6.  }

7.  int main(void)
8.  {   int n;

9.      printf("Nhap n:"); scanf("%d", &n);
10.     printf("n! = %d! = %l\n", n, factorial(n));
11.     return 0;
12. }
```

# Lời gọi hàm đệ quy và Điều kiện dừng của thuật giải đệ quy

- Bài toán giải bằng thuật giải đệ quy phải có điều kiện dừng.

- Thuật toán đệ quy trên máy tính có thể bị giới hạn bởi dung lượng bộ nhớ do lời gọi hàm liên tiếp.

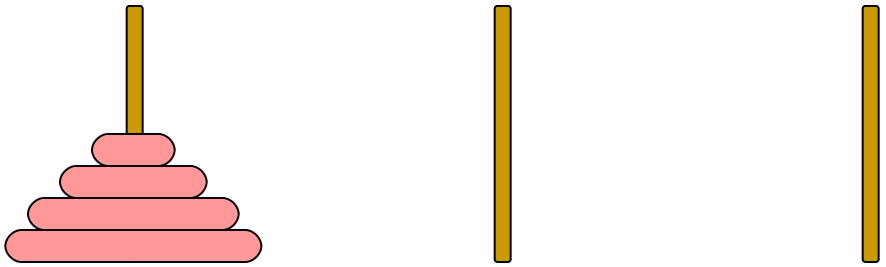



*Hãy vẽ sơ đồ tiến trình gọi hàm khi thực hiện tính dãy fibonacci bằng đệ quy.*

# Bài toán Tháp Hà Nội

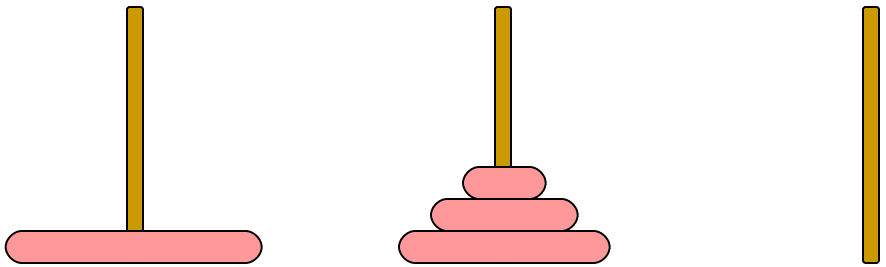
- Có 3 cái cột và một chồng đĩa ở cột thứ nhất. Hãy chuyển chồng đĩa sang cột thứ ba với điều kiện mỗi lần di chuyển chỉ một đĩa và các đĩa bé luôn nằm trên đĩa lớn.

- Truyền thuyết: lúc thế giới hình thành, trong ngôi đền thờ Brahma có một chồng 64 cái đĩa. Mỗi ngày, có một thầy tu di chuyển một đĩa. Đến khi hết đĩa thì đó là ngày tận thế.



# Thuật giải

- Chuyển (n-1) đĩa sang cột trung gian.
- Chuyển đĩa lớn nhất sang cột đích.
- Chuyển (n-1) đĩa từ cột trung gian sang cột đích.



# Cài đặt bằng đệ quy

```
1.  MoveDisk(disk_number, starting_post, target_post,
                                                    intermediate_post)
2.  {
3.      if(disk)number > 1)
4.          {
5.          MoveDisk(disk_number-1, starting_post,
                                    intermediate_post, target_post);
6.          printf("Move disk number %d, from post %d to post %d.\n",
                                    disk_number, starting_post, target_post);
7.          MoveDisk(disk_number-1,intermediate_post,
                                    target_post, starting_post);
8.          }
9.      else
10.         printf("Move disk number 1 from post %d to post %d.\n",
                                            starting_post, target_post);
11. }
```

# Lập chương trình cho máy tính

## CON TRỎ

Lê Hà Thanh
Học kỳ 2, 2004-2005

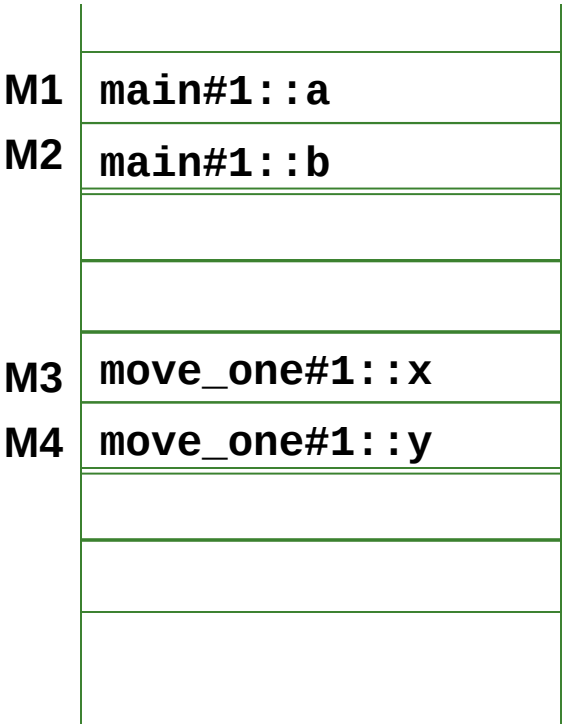
# Con trỏ

- Biến con trỏ
- Khai báo biến con trỏ
- Địa chỉ và giá trị
- Truyền tham chiếu trong lời gọi hàm

# Truyền tham số qua trị

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

2.  void move_one(int x, int y)
3.      {
4.      x = x-1;
5.      y = y+1;
6.      }

7.  int main(void)
8.      {
9.      int a = 4, b = 7;
10.     move_one(a, b);
11.     print("%d, %d\n", a, b);
12.     return 0;
13.     }
```

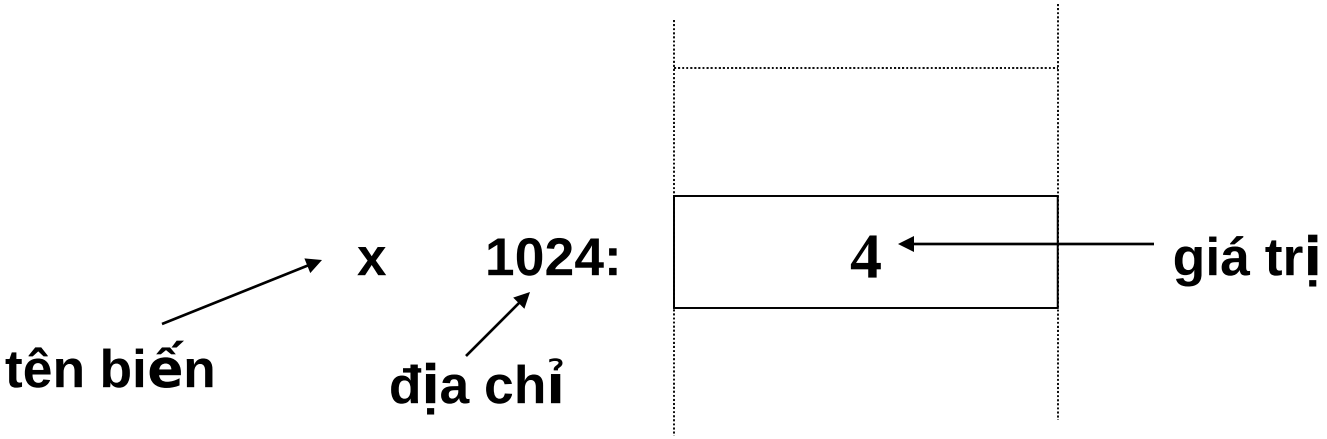
# Bộ nhớ



# Giá trị biến và địa chỉ trong bộ nhớ

- Biến là tên các vùng nhớ được dùng để giữ các giá trị.
- Hàm move_one(a, b) cần truy cập vào các vị trí nhớ của a và b cũng như các giá trị của a và b.

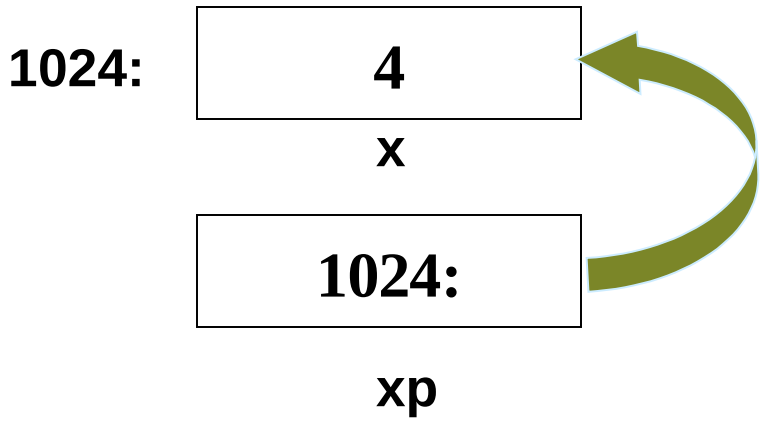
- Bằng cách nào?



# Kiểu dữ liệu Con trỏ

- Một biến kiểu con trỏ (pointer) chứa một tham chiếu (reference) đến một biến loại khác. Nói khác đi, biến con trỏ chứa địa chỉ ô nhớ của một biến.

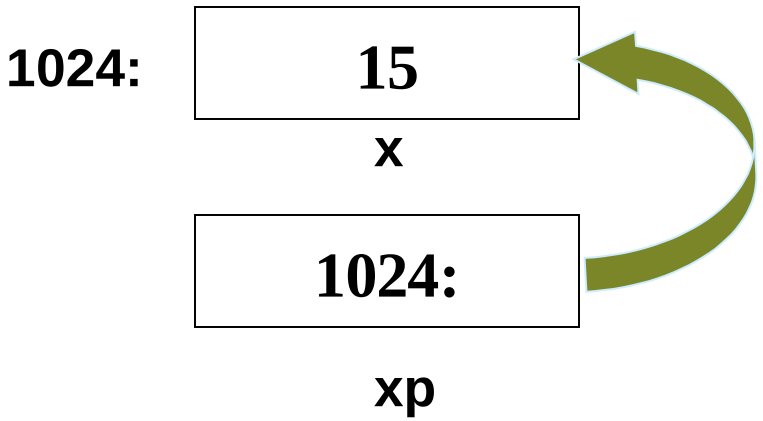
```
int x;
int* xp;  /* con tro tro toi mot so nguyen */

x = 4;
xp = &x;
```



# Sử dụng Con trỏ

- truy cập vùng nhớ được chỉ bởi một con trỏ

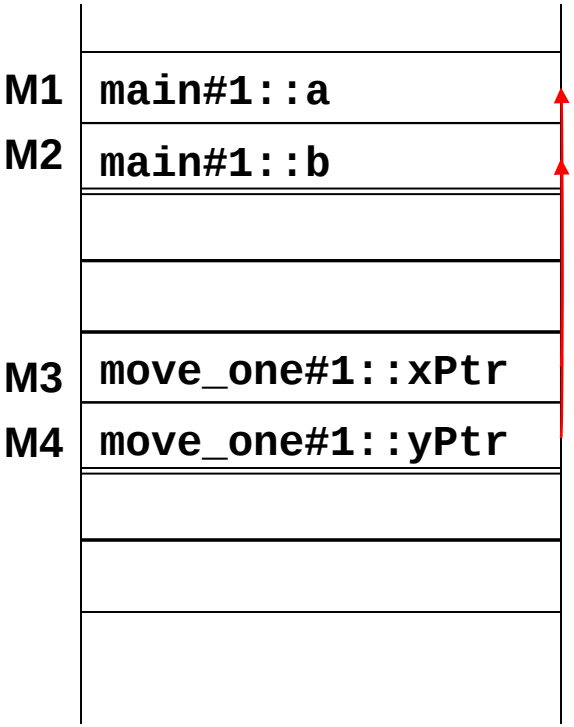
```
xp = &x;      /* gán địa chỉ x vào xp */
*xp = 15;     /* gán giá trị 15 vào biến x */
*xp = *xp + 1;          /* cộng 1 vào x */
```



# Sử dụng con trỏ như tham số

- #include <stdio.h>
- void move_one(int* xPtr, int* yPtr)
  {
  *xPtr = *xPtr-1;
  *yPtr = *yPtr+1;
  }
- int main(void)
  {
  int a, b;
  a=4; b=7;
  move_one(&a, &b);
  print(“%d, %d\n”, a, b);
  return 0;
  }

# Bộ nhớ



# Khai báo, toán tử và sử dụng trong hàm

- Khai báo kiểu dữ liệu con trỏ:
  int *  "con trỏ đến kiểu int"
  float *  "con trỏ đến kiểu float"
  char *  "con trỏ đến kiểu character"

- Toán tử
  & địa chỉ của một đối tượng
  * giá trị của vùng nhớ biến con trỏ chỉ đến

- Con trỏ được dùng như tham số hình thức trong khai báo hàm để truyền và lấy các đối số có giá trị thay đổi.

# scanf

- int x, y;
- printf(“%d %d %d”, x, y, x+y);
- Sử dụng hàm scanf
- scanf(“%d %d %d”, x, y, x+y); /* ??? */
- scanf(“%d %d”, &x, &y);

# Sử dụng Con trỏ

- để lấy các giá trị kết xuất của một hàm.
  ví dụ: hàm move_one(…)
- để lấy nhiều giá trị “trả về” từ một hàm.
  ví dụ: hàm scanf()
- tạo các cấu trúc dữ liệu động.

# Hàm swap

- void swap(int *px, int *py)
  {
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
  }

- main(void)
  {
  int a, b;
  a=2; b=9;
  swap(&a, &b);
  }

# Lập chương trình cho máy tính

## MẢNG

Lê Hà Thanh
Học kỳ 2, 2004-2005

# Hàm swap

- void swap(int *px, int *py)
  {
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
  }

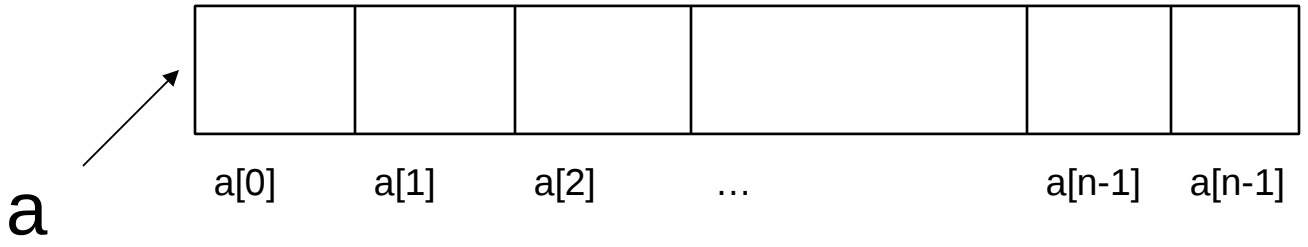
- main(void)
  {
  int a, b;
  a=2; b=9;
  swap(&a, &b);
  }

# Mảng

- Mảng là tập hợp các giá trị cùng kiểu.
- Khai báo:
  typename arrayname[array_size];

- số phần tử trong mảng: array_size;

  int a[array_size];
  n = array_size;




- Truy cập phần tử mảng qua chỉ số của phần tử: i
  array[i]; // 0 <= i <= array_size-1, i ∈ N0

# Chú ý

- C không kiểm tra giới hạn của chỉ số truy cập phần tử mảng. Truy cập đến phần tử i>=array_size không có cảnh báo, nhưng giá trị không kiểm soát được.

- Kích thước mảng phải là một hằng số.

- Kích thước mảng có thể được khai báo tường minh hoặc thông qua một giá trị định nghĩa trước (#define)

  Các khai báo sau đây là hợp lệ

```
int Squares[5] = {0,1,4,9,16};
int Squares[5] = {0,1,4};
int Squares[] = {0,1,4,9,16};
int Squares[];
```

# Chú ý

- Không thể thực hiện các thao tác chép nội dung một mảng sang mảng khác.
  Chép từng phần tử mảng

```
char A[3]={‘a’,’b’,’c’};
char B[3];
B = A; // ???
for(int i=0; i<3; i++)
        B[i] = A[i];
```

  hoặc chép khối bộ nhớ (sẽ được đề cập sau)
- Không dùng phép so sánh trực tiếp (==) nội dung trong hai mảng.
  Phép so sánh (A==B) so sánh địa chỉ hai vùng nhớ mà A và B chỉ đến.

# Chú ý

- Các phần tử trong mảng được dùng như các biến đơn thông thường.

```
1. // nhap gia tri cho cac phan tu mang
2. float a[4];
3. for(int i=0; i<4; i++)
4.         {
5.         printf(“a[%d]=“,i);
6.         scanf(“%f”, &a[i]);
7.         }
```

- Chỉ số của phần tử mảng phải thuộc kiểu nguyên (int, short int, long int, char)

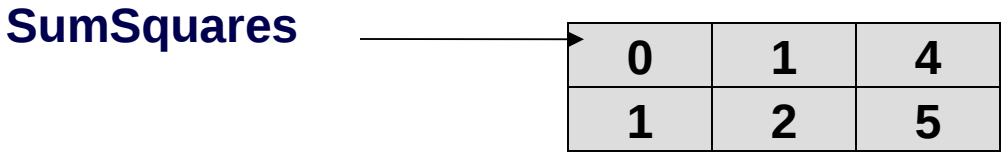
# Mảng trong hàm

```
1.  #include <stdio.h>
2.  #define SIZE 5

3.  void getArray(int *a, int size);

4.  main()
5.      {int an_array[SIZE];
6.      getArray(an_array, SIZE);
7.      return 0;
8.      }

9.  void getArray(int *a, int size)
10.     {for(int i=0; i<size; i++)    {
11.         printf("a[%d]=");
12.         scanf("%d", &a[i]);
13.         }
14.     }
```

# Mảng nhiều chiều

- Khai báo mảng 2 chiều:

  type name[row_size][column_size];




| 0 | 1 | 4 | 1 | 2 | 5 |
|---|---|---|---|---|---|

- int SumSquares[2][3] = { {0,1,4}, {1,2,5} };
- int SumSquares[2][3] = { 0,1,4,1,2,5 };
- int SumSquares[2][3] = { {0,1,4} };
- int SumSquares[ ][3] = { {0,1,4}, {1,2,5} };
- int SumSquares[ ][3] = { {0,1, }, {1} };
- int SumSquares[ ][3];

# Nhập Mảng 2 chiều

```
1.  #define MAX_STUDENT 5
2.  #define MAX_SUBJECT 6

3.  int StudentScore[MAX_STUDENT][MAX_SUBJECT];

4.  void read_Score(int Score[MAX_STUDENT][MAX_SUBJECT],
                                int nStudents, int nSubjects)
5.      {
6.      int i,j;
7.      for(i=0; i<nStudents; i++)
8.          for(j=0; j<nSubjects; j++)
9.              scanf("%d", &Score[i][j]);
10.     }
```

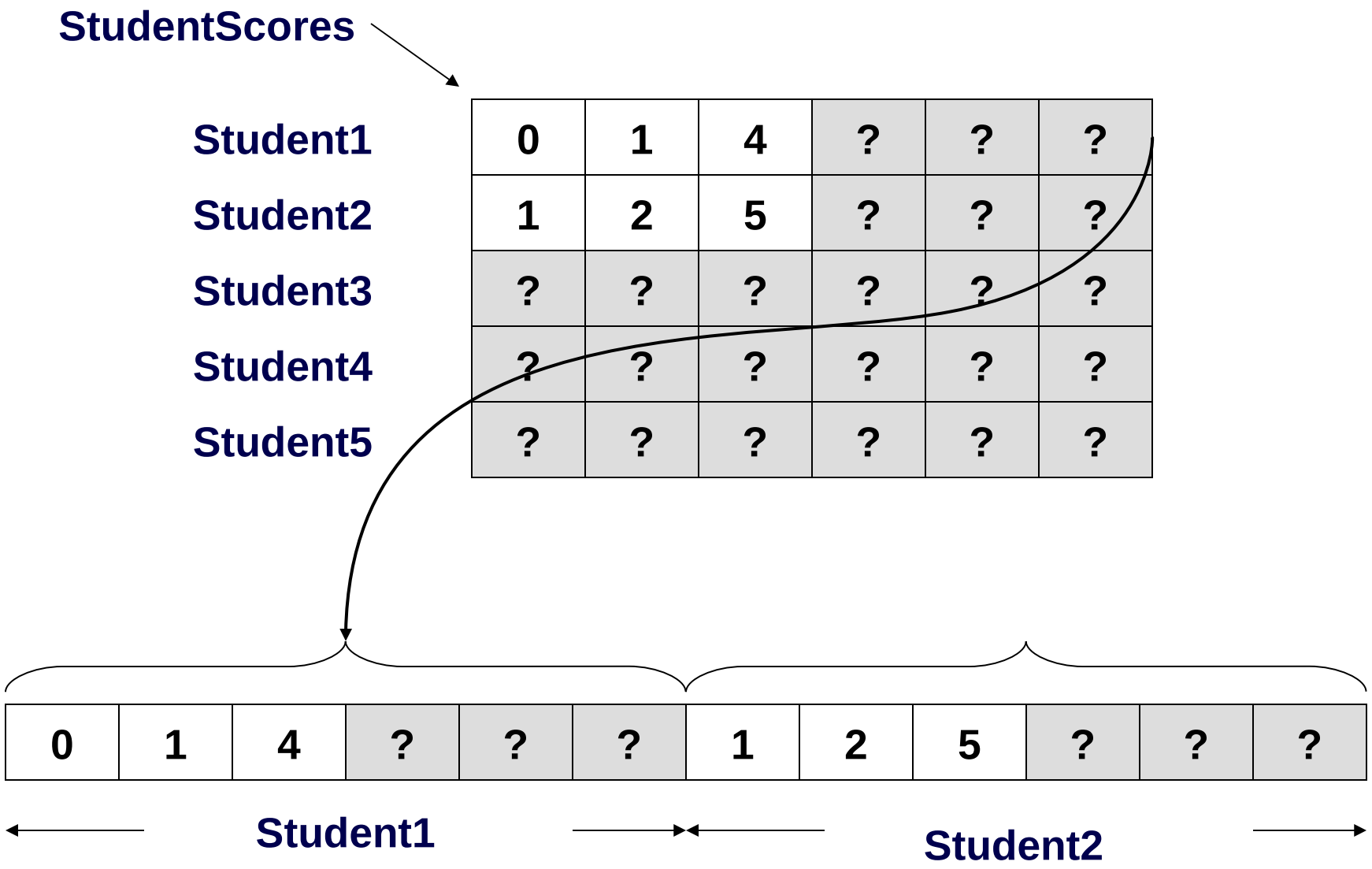
# Hàm truy cập, in Mảng 2 chiều

```
1.  void print_Score(int Score[MAX_STUDENT][MAX_SUBJECT],
                                        int nStudents, int nSubjects)
2.      {
3.      int i,j;

4.      for(i=0; i<nStudents; i++)
5.          {
6.          for(j=0; j<nSubjects; j++)
7.              printf("%2d\t", &Score[i][j]);
8.          printf("\n");
9.          }
10.     }
```

# Chương trình

```
1.  main(void)
2.      {
3.      int nStudents, nScores;

4.      scanf("%d %d", &nStudents, &nScores);
5.      if(nStudents <= MAX_STUDENT &&
                            nScores <= MAX_SCORES)
6.          read_Score(StudentScore, nStudents, nScores);

7.      print_Score (StudentScore, nStudents, nScores);

8.      return 0;
9.      }
```

# Biểu diễn mảng 2 chiều



# Lập chương trình cho máy tính

## KÝ TỰ VÀ CHUỖI KÝ TỰ

Lê Hà Thanh
Học kỳ 2, 2004-2005

# Ký tự (character)

- Kiểu char:
  - ký tự “in được” gồm: 26 chữ thường (a..z), 26 chữ hoa (A..Z), 10 chữ số (0..9), khoảng trắng, các ký tự:
    **! “ # $ % & ‘ ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ { | } ~**
  - Các ký tự “không in được”: tab, lert (bell), newline, formfeed,...

- các ký tự “in được” đặc biệt: ‘\\’, ‘\’’, ‘\”’

- các ký tự “không in được” đặc biệt:
  \n new line
  \a bell
  \0 null character
  \b backspace
  \t horizontal tab
  ...

# Nhập xuất Ký tự

- scanf

```
char ch;
scanf("%c", &ch);
```

- sử dụng các đoạn macro có trong thư viện <stdio.h>

putchar: đưa ký tự ra thiết bị xuất chuẩn (stdout)

```
putchar('\n');
```

getchar: lấy ký tự từ thiết bị nhập chuẩn (stdin)

```
ch = getchar();
```

- getch: lấy trực tiếp ký tự từ bàn phím không hiển thị ra màn hình

```
ch = getch();
```

getche(): lấy trực tiếp ký tự từ bàn phím và hiển thị ký tự ra màn hình.

```
ch = getche();
```

# getchar

```
1.  #include <stdio.h>
2.  int main(void)
3.  {
4.     int c;
5.  /* Note that getchar reads from stdin and is line buffered; this
    means it will not return until you press ENTER. */
6.     while ((c = getchar()) != '\n')
7.        printf("%c", c);
8.     return 0;
9.  }
```

# putchar

```
1.  /* putchar example */
2.  #include <stdio.h>

3.  /* define some box-drawing characters */
4.  #define LEFT_TOP  0xDA
5.  #define RIGHT_TOP 0xBF
6.  #define HORIZ     0xC4
7.  #define VERT      0xB3
8.  #define LEFT_BOT  0xC0
9.  #define RIGHT_BOT 0xD9

10. int main(void)
11. {
12.    char i, j;
13. /* draw the top of the box */
14.    putchar(LEFT_TOP);
15.         for (i=0; i<10; i++)
16.       putchar(HORIZ);
```

```
17.             putchar(RIGHT_TOP);
18.             putchar('\n');

19.     /* draw the middle */
20.       for (i=0; i<4; i++)
21.       {
22.          putchar(VERT);
23.          for (j=0; j<10; j++)
24.             putchar(' ');
25.          putchar(VERT);

26.          putchar('\n');
27.       }
28.     /* draw the bottom */
29.       putchar(LEFT_BOT);
30.       for (i=0; i<10; i++)
31.          putchar(HORIZ);
32.       putchar(RIGHT_BOT);
33.       putchar('\n');

34.       return 0;
35.   }
```

# Một số hàm khác

- kbhit: kiểm tra có phím bấm

```
1. /* khbit example */
2. #include <conio.h>

3. int main(void)
4.     {
5.     cprintf("Press any key to continue:");
6.     while (!kbhit()) /* do nothing */ ;
7.     cprintf("\r\nA key was pressed...\r\n");
8.     return 0;
9.     }
```

# Chuỗi ký tự (string)

- ChuỗI ký tự
- Khai báo biến kiểu chuỗI ký tự.
- Làm việc vớI các biến kiểu chuỗI ký tự.

# Ký tự và Chuỗi

- Các hằng ký tự
  ‘s’, ‘N’, ‘9’, ‘%’, ‘\n’, ‘ ‘, ‘\0’, …

  ‘\0’: ký tự null
- Các hằng chuỗi ký tự:
  “So %d khong la so nguyen to.\n”

- Các biến kiểu ký tự:

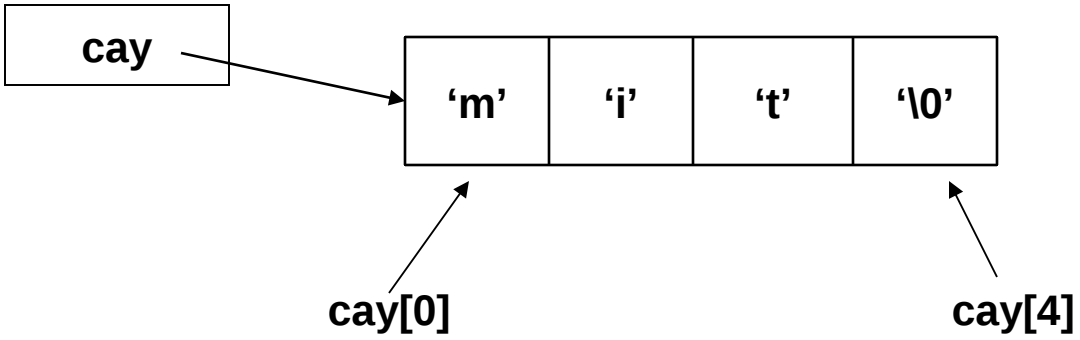
  char ch1=‘c’, ch2=‘n’, ch3=‘t’;
  printf(“Khoa %c%c%c%c\n”, ch1, ch2, ch3, ch3);

# Chuỗi

- Một chuỗi là một mảng một chiều các ký tự,

```
char cay[6]={‘m’,’i’,’t’,’\0’};
printf(“Trong vuon co cay %s\n”, cay);
```

- chính xác hơn chuỗi là mảng một chiều các ký tự kết thúc bằng ký tự kết thúc (null-terminated array of char)



# Khởi tạo một chuỗi

- char ten[10]={‘h’,’o’,’a’,’h’,’o’,’n’,’g’,’\0’};
- char ten[10];
  ten[0]=‘h’; ten[1]=‘o’; ten[2]=‘a’;
  ten[3]=‘h’; ten[4]=‘o’; ten[5]=‘n’; ten[6]=‘g’;
  ten[7]=\0’;
- char ten[10]=“hoahong”;
- char ten[]=“hoahong”;

# Lỗi khi tạo một chuỗi

- Chú ý: không có phép gán trong kiểu dữ liệu chuỗi
- như thế này là sai

```
char ten[10];
ten = “hoahong”;
```

# Chú ý

- Không :
  sử dụng toán tử gán = để chép nội dung của một chuỗi sang chuỗi khác.

```
char  a[4]="hi";
char b[4];
b = a; //???
```

- Không:
  dùng toán tử == để so sánh nội dung hai chuỗi

```
char a[] = "hi";
char b[] = "there";
if(a==b) //???
   { }
```

# Làm việc với chuỗi

- #include <string.h>

- Với hàm scanf và printf

```
1. #include <stdio.h>
2. void main (void)
3.     {
4.     char name[20];
5.     printf (“Enter a name: “);
6.     scanf (“%s”, name); // there is no & before name
7.     printf (“Hello %s\n”, name);
8.     }
```

# strlen()

- size_t strlen(const char *s);

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

3. int main(void)
4.     {
5.     char *string = "Borland International";
6.     printf("%d\n", strlen(string));      // 21
7.     return 0;
8.     }
```

# Gán một chuỗi vào chuỗi khác

- gán từng ký tự trong chuỗi.

```
1.      char str[] = “qua mang cut”;
2.      char newstr[30];
3.      for(int i = 0; i<strlen(str); i++)
4.              newstr[i] = str[i];
5.      newstr[i] = ‘\0’;
```

- dùng hàm strcpy trong <string.h>

```
char *strcpy(char *dest, const char *src);
```

chép chuỗi src sang chuỗi dest cho đến khi gặp null character

```
strcpy(newstr, str);
```

# Ghép chuỗi bằng strcat()

- char *strcat(char *dest, const char *src);

- Ghép chuỗi src vào cuối chuỗi dest. Chiều dài chuỗi kết quả là strlen(dest) + strlen(src)

1. char destination[25];
2. char *blank = " ", *c = “C++“;
3. char *turbo = "Turbo";

4. strcpy(destination, turbo);
5. strcat(destination, blank);
6. strcat(destination, c);
7. printf("%s\n", destination);
8. ...

# So sánh chuỗi: strcmp()

- int strcmp(const char *s1, const char*s2);

- kết quả: số âm nếu s1 < s2
  0 nếu s1 == s2
  số dương nếu s1 > s2

- Hàm strncmp()

  int strncmp(const char *s1, const char *s2, size_t maxlen);

  so sánh hai chuỗi với chiều dài cần so sánh là maxlen

# strncat() và strncopy()

- strncpy
  char *strncpy(char *dest, const char *src,  size_t maxlen);

  chép tối đa maxlen ký tự từ chuỗi src sang chuỗi dest

- strncat
  char *strncat(char *dest, const char *src,  size_t maxlen);

  ghép tối đa maxlen ký tự trong chuỗi src vào cuối chuỗi dest

# Biến đổi chuỗi sang số

- atoi(), atof(), atol():
  đổi chuỗi ký tự sang số.

```
int atoi(const char *s);
double atof(const char *s);
long atol(const char *s);
```

- ...

```
float f;
char *str = "12345.67";

f = atof(str);
printf("string = %s float = %f\n", str, f);
...
```

# Nhập/xuất chuỗi

- gets: lấy chuỗi ký tự từ thiết bị nhập chuẩn stdin
  puts: đưa chuỗi ký tự ra thiết bị xuất chuẩn stdout

```
char *gets(char *s);
int puts(const char *s);
```

- vd:

```
1.  char string[80];

2.  printf("Input a string:");
3.  gets(string);
4.  printf("The string input was: %s\n", string);
5.  puts(string);
```

# Kiểu liệt kê (enumerated type)

- Kiểu liệt kê khai báo một tập các hằng số theo thứ tự.
- Nếu không có khai báo rõ ràng, các hằng bắt đầu từ giá trị 0.

```
enum NgayTrongTuan {Hai, Ba, Tu, Nam, Sau, Bay, CN};

enum NgayTrongTuan ngay;
ngay = Hai; // 0
```

- Khai báo giá trị khác cho các hằng.

```
enum Month {Jan = 1, Feb = 2, Mar, Apr, Jun, July};
```

# Lập chương trình cho máy tính

## CẤP PHÁT BỘ NHỚ ĐỘNG

Lê Hà Thanh
Học kỳ 2, 2004-2005

# Cấp phát động: malloc() và calloc()

- Hàm malloc và calloc cho phép cấp phát các vùng nhớ ngay trong lúc chạy chương trình.

  void *malloc( size_t size);

  void *calloc( size_t nItems, size_t size);

- Hàm calloc cấp phát vùng nhớ và khởi tạo tất cả các bit trong vùng nhớ mới cấp phát về 0.

- Hàm malloc chỉ cấp phát vùng nhớ.

# Ví dụ 1: dùng malloc()

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  #include <alloc.h>
4.  #include <process.h>

5.  int main(void)
6.  {   char *str;
7.      /* allocate memory for string */
8.      if ((str = (char *) malloc(10)) == NULL)
9.         {
10.        printf("Not enough memory to allocate buffer\n");
11.        exit(1);  /* terminate program if out of memory */
12.        }
```

# Ví dụ 1: (tt)

```
13.         /* copy "Hello" into string */
14.     strcpy(str, "Hello");

15.     /* display string */
16.     printf("String is %s\n", str);

17.     /* free memory */
18.     free(str);
19.     return 0;
20. }
```

# Ví dụ 2: calloc()

```
1.  #include <stdio.h>
2.  #include <alloc.h>
3.  #include <string.h>

4.  int main(void)
5.  {
6.      char *str = NULL;

7.      /* allocate memory for string */
8.      str = (char *) calloc(10, sizeof(char));

9.      /* copy "Hello" into string */
10.     strcpy(str, "Hello");
```

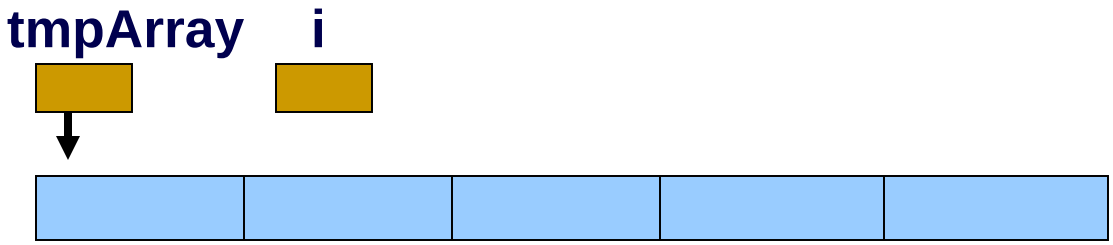
# Ví dụ 2: calloc()

```
11.     /* display string */
12.     printf("String is %s\n", str);

13.     /* free memory */
14.     free(str);

15.     return 0;
16. }
```

# Giải phóng vùng nhớ

- Khi thoát khỏi hàm, các biến khai báo trong hàm sẽ “biến mất”. Tuy nhiên các vùng nhớ được cấp phát động vẫn còn tồn tại và được “đánh dấu” là đang “được dùng” → bộ nhớ của máy tính sẽ hết.
- ví dụ:

```
void aFunction(void)
{
        int *tmpArray, i = 5;

        tmpArray = (int *)malloc(i * sizeof(int));
}
```



# Giải phóng vùng nhớ: free

- Sử dụng các cặp hàm

  (malloc, free)
  (calloc, free)

- C dùng hàm free để giải phóng vùng nhớ cấp phát động.

  void free(void *block);

# Cấp phát lại vùng nhớ: realloc

- Đôi khi chúng ta muốn mở rộng hoặc giảm bớt kích thước mảng.
- C dùng hàm realloc để cấp phát lại vùng nhớ, thực hiện chép nội dung của vùng nhớ cũ sang vùng nhớ mới.

  void *realloc(void *block, size_t size);
- ví dụ:

```
int *tmpArray, N=5,i;

tmpArray = (int *)malloc(N * sizeof(int));
for(i = 0; i<N; i++)
        tmpArray]i] = i;
tmpArray = (int *)realloc(tmpArray, 7);
```

# Ví dụ: realloc()

```
1.  #include <stdio.h>
2.  #include <alloc.h>
3.  #include <string.h>

4.  int main(void)
5.  {
6.     char *str;

7.     /* allocate memory for string */
8.     str = (char *) malloc(10);
```

# Ví dụ: realloc() - tt

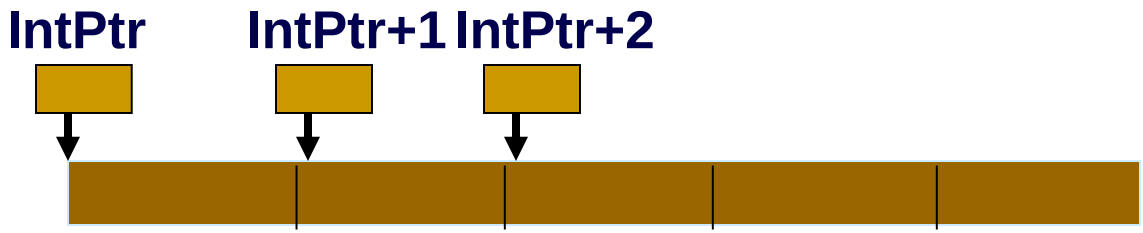
```
9.  /* copy "Hello" into string */
10.    strcpy(str, "Hello");

11.    printf("String is %s\n  Address is %p\n", str, str);
12.    str = (char *) realloc(str, 20);
13.    printf("String is %s\n  New address is %p\n", str, str);

14.    /* free memory */
15.    free(str);

16.    return 0;
17. }
```

# Di chuyển con trỏ trong mảng

```
int IntArr[5] = {3,4,2,1,0};
char CharArr[5] = {‘a’,’b’,’c’,’d’,’e’};
int *IntPtr = IntArr;
char *charPtr = CharArr;
```

- Di chuyển trong mảng 1 chiều:

```
int val_at_0 = * IntPtr;          // = IntArr[0]
int val_at_3 = * (IntPtr+3); // = IntArr[3]
```



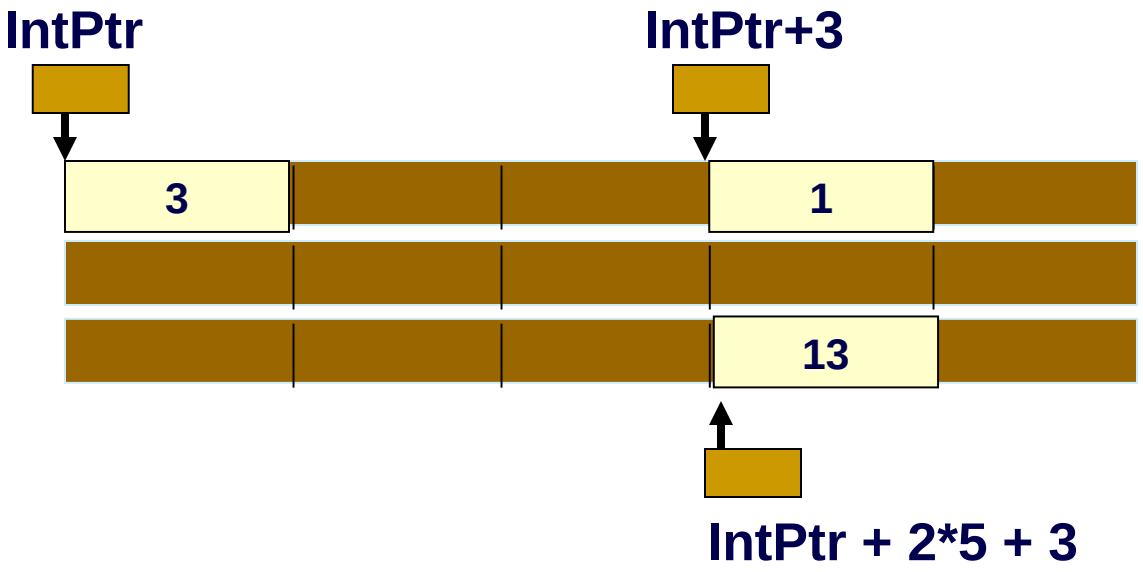
# Phép toán với con trỏ trong mảng

```
#define ROWS     5
#define COLS      3
int IntArr[ROWS][COLS] = {{3,4,2,1,0},
                          {5,6,7,8,9},
                          {10,11,12,13,14}};
int *IntPtr = IntArr;
```

- Tham chiếu đến phần tử trong mảng 2 chiều:

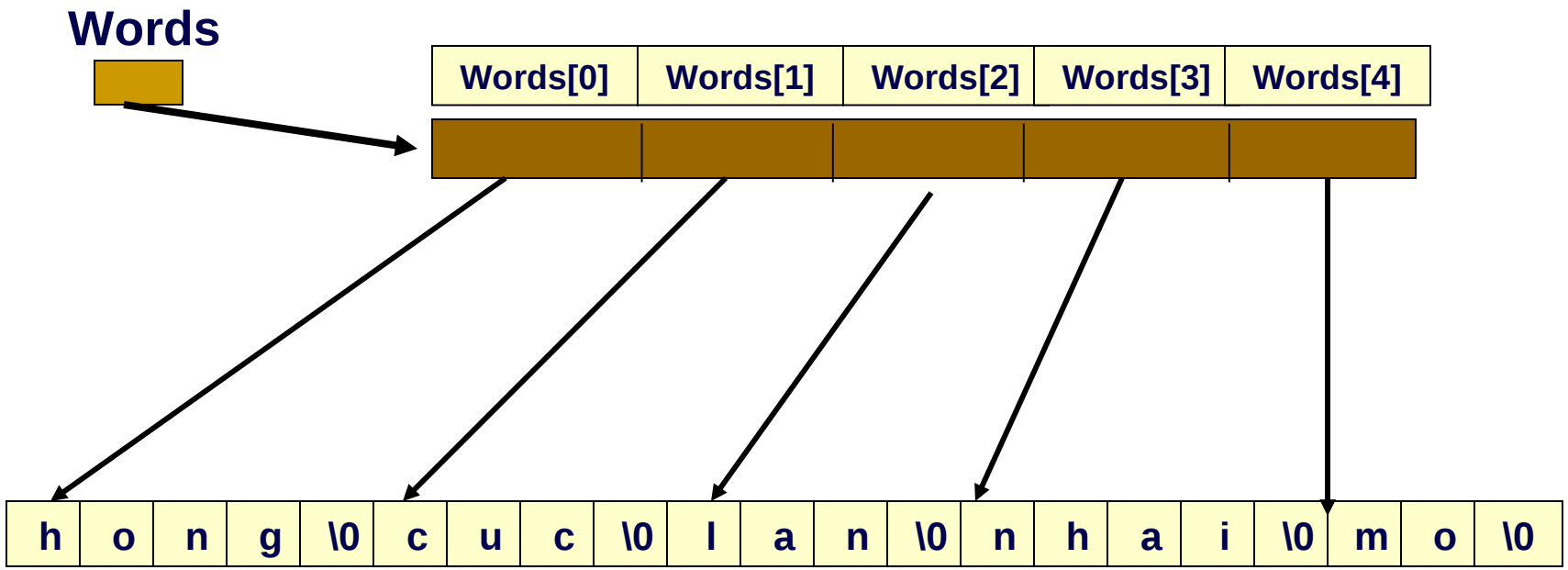
```
int val_at_00 = * IntPtr;// = IntArr[0][0]
int val_at_30 = * (IntPtr+3);// = IntArr[3][0]
int val_at_32 = * (intPtr+3*5+2);// = IntArr[3][2]
int val_at_32 = * ( * (IntPtr+3)
```

# Phép toán với con trỏ trong mảng



# Mảng các chuỗi: char * [ ]

- char *Words[ ] = {“hong”, “cuc”, “lan”, “nhai”,”mo”};
-



# Lập chương trình cho máy tính

## KIỂU CẤU TRÚC (STRUCTURE)

Lê Hà Thanh
Học kỳ 2, 2004-2005

# Cấu trúc (structure)

- Cấu trúc dùng lưu tập hợp các đối tượng không cùng kiểu.

- Khai báo:

```
struct <<structure tag name>>
{
        << type >> << field1 >>  ;
        << type >> << field2 >>  ;
        << type >> << field3 >>  ;
        << ... >>
}
```

# Định nghĩa Cấu trúc

```
1. struct SinhVien
2.     {
3.     char       *hoten; // toi da 30 ky tu
4.     int                namsinh;        // >= 1960
5.     char       *noisinh;          // 3 chu cai viet tat cua noi sinh
6.     char       *maso; // maso dai toi da 10 ky tu
7.     }

8. // khai bao kieu SinhVien
9. typedef struct SinhVien SinhVien;
```

# Kích thước của một cấu trúc

- Kích thước kiểu dữ liệu cấu trúc:

  sizeof(<<typename>>)

- vd:

  SinhVien_t_size = sizeof(SinhVien);  // 48

# Khai báo biến cấu trúc

1. struct SinhVien s301160101;
2. SinhVien s301160102;

3. SinhVien c01vta1[SoSinhVien];

4. SinhVien X = ( “Nguyen Van X”, 1983, “HN”, “301160112” );
5. SinhVien Y = s301160102;

# Khai báo biến cấu trúc (tt)

```
1. struct SinhVien
2.     {
3.     char        hoten[31]; // toi da 30 ky tu
4.     int         namsinh;   // >= 1960
5.     char        noisinh[4];// 3 chu cai viet tat cua noi sinh
6.     char        maso[11];  // maso dai toi da 10 ky tu
7.     } X, Y, Z;

8. struct SinhVien NguyenLe = (“Nguyen Le”, 1983, “HU”,
                                “301160123”);

9. Z = NguyenLe;
```

# Chú ý

- Khi khai báo biến cấu trúc, nếu
  - khai báo nhiều giá trị khởi tạo hơn số trường của kiểu dữ liệu → sai.
  - khai báo giá trị khởi tạo cho một số trường trong cấu trúc, các trường còn lại sẽ tự động được gán giá trị 0.

# Kiểu Cấu trúc (structure type)

- Định nghĩa kiểu:

```
typedef struct
        {
                << type >> << field1 >>  ;
                << type >> << field2 >>  ;
                << type >> << field3 >>  ;
                << ... >>
        } << type name >>;
```

# Truy cập thành phần trong Cấu trúc

- Toán tử thành viên: .

```
X.hoten          // “Nguyen Van X”
X.namsinh        // 1983

gets(X.hoten);
scanf(“%d %s %s”, &X.namsinh, X.noisinh,
                                  X.maso);
```

- Có thể thực hiện phép gán biến cấu trúc này sang biến cấu trúc khác.

```
X = NguyenLe;
```

Thực chất đây là phép gán từng bit.

- Tuy nhiên với các cấu trúc có mảng ở trong, nên dùng cách chép từng thành phần.

# Mảng các Cấu trúc

- Khai bao biến mảng các Cấu trúc cũng giống như khai báo biến mảng trên các kiểu dữ liệu cơ bản khác.

- Dùng tên mảng khi truy cập từng phần tử trong mảng các cấu trúc và toán tử thành viên để truy cập các trường dữ liệu của từng thành phần cấu trúc.

# Mảng các Cấu trúc (tt)

```
1.  int i = 0;
2.  int tieptuc = 1;

3.  while(tieptuc)
4.      {
5.      gets(c01vta1[i].hoten);
6.      scanf("%d %s %s", &c01vta1[i].namsinh,
                c01vta1[i].noisinh, c01vta1[i].maso);
7.      printf("\nTiep tuc? (C/K)");
8.      tieptuc = (toupper(getch()) == 'C' ? 1 : 0);
9.      i++;
10.     }
```

# Con trỏ đến Cấu trúc

- Khai báo con trỏ đến cấu trúc tương tự như các kiểu dữ liệu khác.

```
SinhVien *SV_ptr;
```

- Toán tử truy cập trường thành phần của cấu trúc do con trỏ chỉ đến: ->

```
scanf("%d %s", &SV_ptr->namsinh, SV_ptr->noisinh);
```

# Union

- Khai báo union được dùng để khai báo các biến dùng chung bộ nhớ.

```
union int_or_long {
        int     i;
        long    l;
} a_number;
```

- Các thành phần của union có thể không có kích thước bằng nhau.

- Kích thước bộ nhớ trong khai báo union là kích thước của kiểu dữ liệu lớn nhất có trong khai báo union.

# Ví dụ

```
1.  union int_or_long {
2.          int         i;
3.          long        l;
4.  } a_number;
5.  a_number.i = 5;
6.  a_number.l = 100L;

7.  // anonymous union
8.  union {
9.          int         i;
10.         float       f;
11.         };
12. i = 10;
13. f = 2.2;
```

# Lập chương trình cho máy tính

## TẬP TIN (FILE)

Lê Hà Thanh
Học kỳ 2, 2004-2005

# Tập tin (FILE)

- Kiểu FILE *
- Tập tin văn bản.
- Tập tin nhị phân.

# Kiểu FILE *

- Kiểu FILE * (khai báo trong stdio.h) cho phép làm việc với các tập tin (văn bản, nhị phân).
- Khai báo con trỏ tập tin

```
FILE * fp;
```

- Chúng ta sử dụng con trỏ tập tin để truy cập (đọc, ghi, thêm thông tin) các tập tin.

# Mở tập tin

FILE * fopen( const char *FileName, const char *Mode);

- Filename: tên tập tin cần mở. Có thể chỉ định một đường dẫn đầy đủ chỉ đến vị trí của tập tin.
- Mode: chế độ mở tập tin: chỉ đọc, để ghi (tạo mới), ghi thêm.
- Nếu thao tác mở thành công, fopen trả về con trỏ FILE trỏ đến tập tin FileName.
- Nếu mở không thành công (FileName không tồn tại, không thể tạo mới), fopen trả về giá trị NULL.

# Đóng tập tin

```
int  fclose( FILE *filestream );
```

- filestream: con trỏ đến tập tin đang mở cần đóng.

- Nếu thao tác đóng thành công, fclose trả về 0.
- Nếu có lỗi (tập tin đang sử dụng), fclose trả về giá trị EOF.

# Ví dụ : Mở, Đóng tập tin

```
1. FILE * fp;

2. // mở VB.TXT “chỉ đọc”
3. if( (fp = fopen( “C:\\LTC\\VB.TXT”, “r” )) == NULL)
4.     {  printf( “Tap tin khong mo duoc\n”);
5.        exit(1);
6.     }

   /* ... */

7. fclose( fp );
```

# Tập tin văn bản

- Tập tin văn bản là kiểu tập tin được lưu trữ các thông tin dưới dạng kiểu ký tự.
- Truy xuất tập tin văn bản:
  - theo từng ký tự
  - theo từng dòng
- Chế độ mở trên tập tin văn bản
  - “r” : đọc (tập tin phải có trên đĩa)
  - “w” : ghi (ghi đè lên tập tin cũ hoặc tạo mới nếu tập tin không có trên đĩa)
  - “a” : ghi nối vào cuối tập tin.
  - “r+” : đọc/ghi. Tập tin phải có trên đĩa.
  - “a+” : đọc, ghi vào cuối tập tin. Tạo mới tập tin nếu tập tin chưa có trên đĩa.

# getc, putc, fgetc, fputc

- getc: nhận ký tự từ tập tin.

```
int getc ( FILE *fp );
```

getc trả về ký tự đọc được hoặc
trả về EOF nếu fp không hợp lệ hoặc đọc đến cuối tập tin.

- putc: ghi ký tự ra tập tin.

```
int putc ( int Ch, FILE * fp );
```

putc trả về EOF nếu thao tác ghi có lỗi.

- có thể dùng fgetc và fputc.

# Ví dụ 1: getc() đọc phím đến khi Enter

```
1.  #include <stdio.h>
2.  #include <conio.h>
3.  #include <stdlib.h>
4.  void main()
5.      { FILE * fp;
6.      char filename[67], ch;
7.      printf ( “Filename: “ );
8.      gets (filename);
9.      if ((fp = fopen (filename, “w” )) == NULL ) // mở tập tin mới để ghi
10.         { printf ( “Create file error \n”); exit (1); }
11.     while (( ch = getche() ) != ‘\r’ ) // đọc cho đến khi gặp ENTER
12.         putc ( fp );
13.     fclose ( fp );
14.     }
```

# Ví dụ 2: putc() in nội dung tập tin văn bản ra màn hình

```
1.  #include <stdio.h>
2.  #include <conio.h>
3.  #include <stdlib.h>
4.  void main()
5.      { FILE * fp;
6.      char filename[67], char ch;
7.      printf ( “Filename: “ );
8.      gets (filename);
9.      if ((fp = fopen (filename, “r” )) == NULL ) // mở tập tin mới để đọc
10.         { printf ( “Open file error \n”); exit (1); }
11.     while (( ch = getc ( fp ) ) != EOF ) // đọc cho đến hết tập tin
12.         printf ( “%c”, ch );
13.     fclose ( fp );
14.     }
```

# Ví dụ 3: Đếm số từ trong tập tin văn bản.

```
1.  #include <stdio.h>
2.  #include <conio.h>
3.  #include <stdlib.h>
4.  void main()
5.      { FILE * fp;
6.      char filename[67], char ch;
7.      int count = 0, isword = 0;

8.      printf ( “Filename: “ );  gets (filename);
9.      if ((fp = fopen (filename, “r” )) == NULL ) // mở tập tin mới để đọc
10.        { printf ( “Open file error \n”); exit (1); }
```

```
11.  while (( ch = getc ( fp ) ) != EOF ) // đọc cho đến hết tập tin
12.      {
13.      if (( ch >= ‘a’ && ch <= ‘z’ ) || ( ch >= ‘A’ && ch <= ‘Z’ ))
14.           isword =1;
15.      if (( ch == ‘ ‘ || ch == ‘\n’ || ch == ‘\t’ ) && isword )
16.          { count ++; isword = 0; }
17.      }

18.  printf ( “Number of word: %d\n”, count);
19.  fclose ( fp );
20.  }
```

# fgets()

- fgets: đọc chuỗi ký tự từ tập tin.

  ```
  char * fgets ( char *Str, int NumOfChar, FILE *fp );
  ```

  fgets đọc các ký tự trong tập tin cho đến khi gặp một trong các điều kiện:
  - EOF
  - gặp dòng mới
  - đọc được (NumOfChar - 1) ký tự trước khi gặp hai điều kiện trên.

- fgets trả về chuỗi ký tự đọc được (kết thúc bằng \0) hoặc trả về con trỏ NULL nếu EOF hoặc có lỗi khi đọc.

# fputs()

- fputs: ghi chuỗi ký tự ra tập tin.

```
int fputs ( const char *Str, FILE * fp );
```

- fputs trả về EOF nếu thao tác ghi có lỗi.

# Hàm chép tập tin văn bản

- Có trong các thư viện

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

- /* Chép từ SourceFile sang DestFile và trả về số ký tự đọc đươc

# feof(); fscanf(), fprintf(); fflush()

- feof: cho biết đã đến cuối tập tin chư a(EOF).

  int feof (FILE *fp );

- fprintf:

  int fprintf ( FILE *fp, const char * Format, ... );

- fscanf:

  int fscanf ( FILE *fp, const char * Format, ...);

- fflush:

  int fflush ( FILE * fp );

  Buộc ghi ra tập tin các dữ liệu có trong buffer.

# Tập tin Nhị phân

- Tập tin nhị phân là một chuỗi các ký tự, không phân biệt ký tự in được hay không in được.
- Tập tin nhị phân thường dùng để lưu trữ các cấu trúc (struct) hoặc union.
- Khai báo:

```
FILE * fp;
```

- Truy xuất tập tin nhị phân theo khối dữ liệu nhị phân.

# Tập tin Nhị phân

- Các chế độ mở tập tin nhị phân:
  - “rb” : mở chỉ đọc
  - “wb” : ghi (ghi đè lên tập tin cũ hoặc tạo mới nếu tập tin không có trên đĩa)
  - “ab” : ghi nối vào cuối tập tin.
  - “rb+” : đọc/ghi. Tập tin phải có trên đĩa.
  - “wb+” : tạo mới tập tin cho phép đọc ghi.
  - “ab+” : đọc, ghi vào cuối tập tin. Tạo mới tập tin nếu tập tin chưa có trên đĩa.

# Đọc ghi tập tin Nhị phân

- fread() : đọc

```
size_t   fread ( void *Ptr, size_t ItemSize,
                                    size_t NumItem, FILE * fp );
```

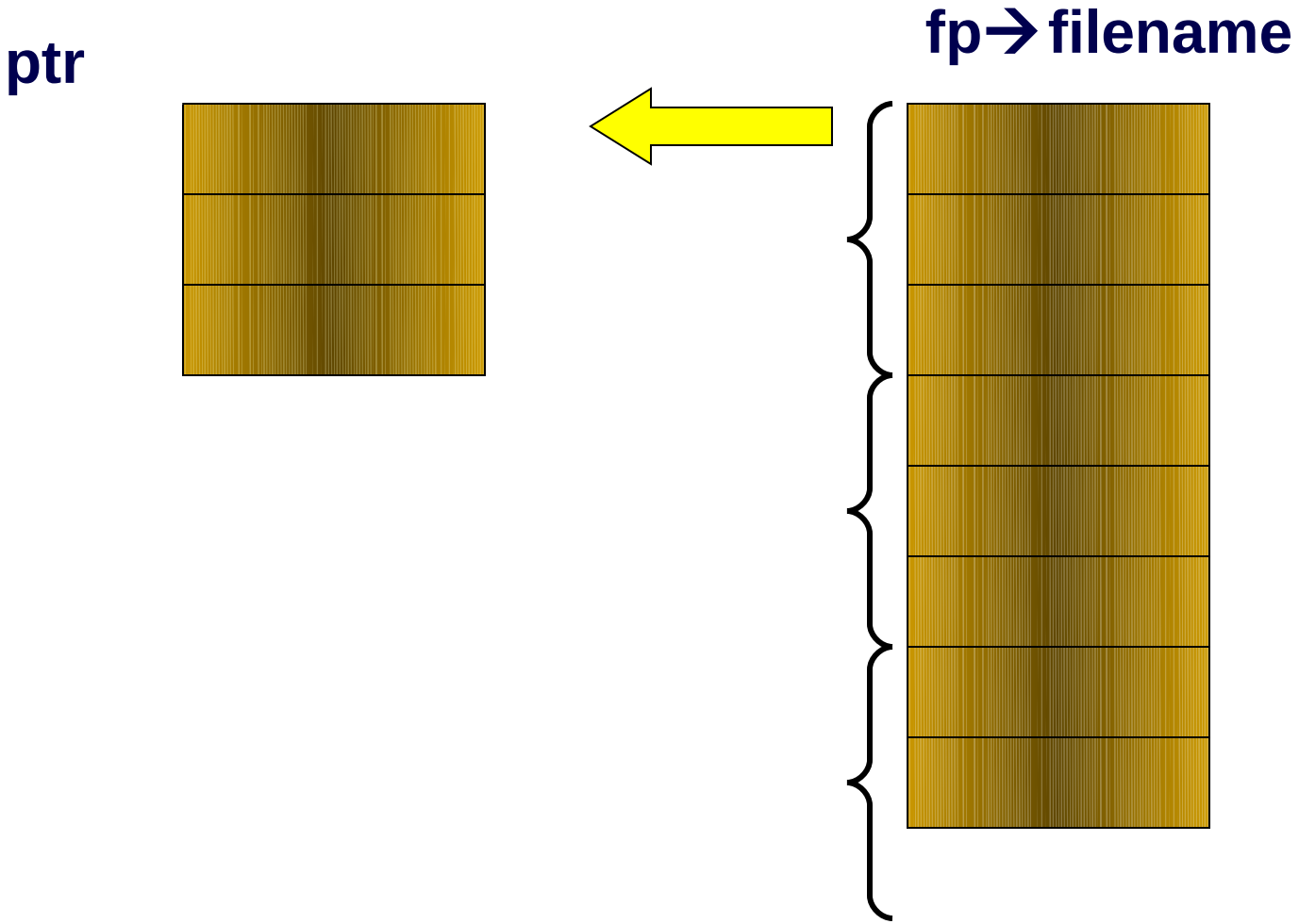
fread đọc NumItem khối dữ liệu, mỗi khối có kích thước ItemSize từ fp và chứa vào vùng nhớ xác định bởi Ptr.
fread trả về số khối dữ liệu đọc được.
Nếu có lỗi hoặc EOF thì giá trị trả về nhỏ hơn NumItem

- fwrite() : ghi

```
size_t  fwrite ( const void *Ptr, size_t ItemSize,
                                    size_t NumItem, FILE * fp );
```

fwrite ghi khối (NumItem x ItemSize) xác dịnh bởi Ptr ra fp.

- Chép 3 mục từ tập tin filename vào vùng nhớ trỏ bởi ptr



# Con trỏ FILE

- Một tập tin sau khi mở được quản lý thông qua một con trỏ FILE.
- Khi mở tập tin (wb, rb), con trỏ FILE chỉ đến đầu tập tin.
- Khi mở tập tin (ab), con trỏ FILE chỉ đến cuối tập tin.
- Con trỏ FILE chỉ đến từng byte trong tập tin nhị phân.
- Sau mỗi lần đọc tập tin, con trỏ FILE sẽ di chuyển đi một số byte bằng kích thước (byte) của khối dữ liệu đọc được.

# fseek(),

- Các hằng dùng trong di chuyển con trỏ FILE

```
#define     SEEK_SET    0
#define     SEEK_CUR    1
#define     SEEK_END    2
```

- **int fseek( FILE *fp, long int offset, int whence );**

  fseek di chuyển con trỏ fp đến vị trí offset theo mốc whence
- offset: khoảng cách (byte) cần di chuyển tính từ vị trí hiện tại. (offset > 0: đi về phía cuối tập tin, offset < 0: ngược về đầu tập tin)
- whence: SEEK_SET: tính từ đầu tập tin
  SEEK_CUR: tính từ vị trí hiện hành của con trỏ
  SEEK_END: tính từ cuối tập tin
- fseek trả về: 0 nếu thành công, <>0 nếu di chuyển có lỗi

# ftell() và rewind()

- rewind đặt lại vị trí con trỏ về đầu tập tin.

```
void rewind ( FILE * fp );
```

tương đương với fseek ( fp, 0L, SEEK_SET);

- ftell trả về vị trí offset hiện tại của con trỏ

```
long int ftell ( FILE * fp );
```

Nếu có lỗi, ftell trả về -1L

# Ví dụ: xác định kích thước tập tin.

- ...// khai báo biến cẩn thận

```
1. if ( fp = fopen ( FileName, “rb” ) ) == NULL )
2.         fprintf( stderr, “Cannot open %s\n”, FileName );
3. else
4.         {
5.         fseek( fp, 0, SEEK_END );
6.         FileSize = ftell( fp );
7.         printf( “File size : %d bytes\n “, FileSize );
8.         fclose( fp );
9.         }
```

# Vị trí con trỏ: fgetpos() và fsetpos()

- với các tập tin có kích thước cực lớn fseek và ftell sẽ bị giới hạn bởi kích thước của offset.

- Dùng

```
int fgetpos ( FILE *fp, fpos_t *position);

int fsetpos ( FILE *fp, const fpos_t *position);
```

# Xóa và đổi tên tập tin

- Thực hiện xoá

  int remove(const char *filename);

- đổi tên tập tin

  int rename(const char *oldname, const char *newname);

# Chú ý khi làm việc với tập tin

- Khi mở tập tin filename, tập tin này phải nằm cùng thư mục của chương trình hoặc

- phải cung cấp đầy đủ đường dẫn đến tập tin

  vd: C:\baitap\taptin.dat
  viết như thế nào?

  fp = fopen( “C:\baitap\taptin.dat”, “rb” );

  SAI RỒI

- vì có ký tự đặc biệt ‘\’ nên viết đúng sẽ là:
  fp = fopen( “C:\\baitap\\taptin.dat”, “rb” );

# Tham số dòng lệnh chương trình

- Khi gọi chạy một chương trình, chúng ta có thể cung cấp các tham số tại dòng lệnh gọi chương trình.
  ví dụ: dir A:*.c  /w
  “copy”, “A:*.c” và “/w” là hai tham số điều khiển chương trình.
- command-line arguments.
- Các tham số dòng lệnh được tham chiếu qua hai đối số khai báo trong hàm main.

```
main ( int argc, char * argv[ ] )
    { ... }
```

- *argc: argument count*
- *argv: argument vector*

# Tham số dòng lệnh – ví dụ

- ... // addint.c → addint.exe

```
1. void main (int argc, char * argv [ ])
2.     {
3.       int res = 0;
4.       printf (“ Program %s\n “, argv[0]);
5.       for ( int i = 1; i < argc; i++ )
6.                 res += atoi(argc[i]);
7.       printf (“ %d\n”, res );
8. }
```

- G:\>addint 3 –2 1 5 6 -2 1
  12

# Lập chương trình cho máy tính

## CẤU TRÚC DỮ LIỆU – STACK và QUEUE

Lê Hà Thanh
Học kỳ 2, 2004-2005

# Cấu trúc dữ liệu

- Stack
- Queue
- Các phép toán trên stack và queue
- Biểu diễn stack và queue bằng mảng

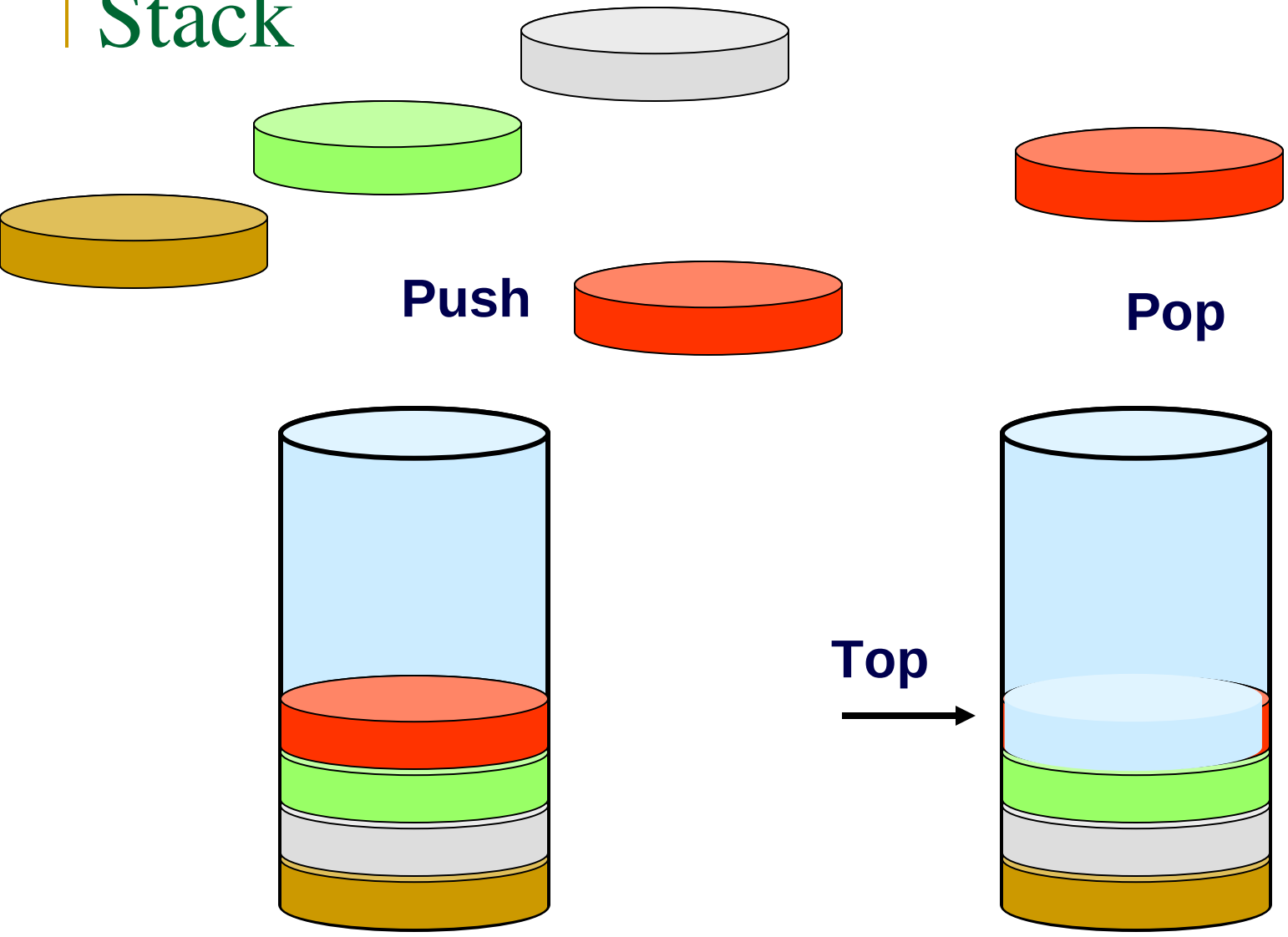
# Stack

- Stack là cấu trúc dữ liệu trong đó các truy xuất đến phần tử trong stack chỉ thực hiện được với phần tử được chèn vào mới nhất

- LIFO: Last In First Out - vào sau ra trước.

- Các thao tác trong stack:
  - push: chèn phần tử mới vào stack
  - pop: lấy phần tử đầu stack ra khỏi stack
  - top: kiểm tra phần tử đầu stack

# Stack



# Khai báo cấu trúc dữ liệu cho stack

```
1. struct StackPtr
2.     {
3.       int *Array;        // mảng các phần tử
4.       int TopOfStack;   // vị trí phần tử đỉnh stack
5.       int MaxSize;    // số phần tử tối đa trong stack
6.     } *Stack;

7. static const StInitSize = 5;
```

# Kiểm tra Stack rỗng: StIsEmpty()

```
1. // ------------ Kiểm tra Stack rỗng ---------------
2. int StIsEmpty (const Stack S )
3.     {
4.     StInsistGood ( S );
5.     return (S→TopOfStack == -1);
6.     }
```

# Kiểm tra Stack tồn tại StInsistGood()

```
1. // ------------------ Kiểm tra Stack có tồn tại ------------------
2. void StInsistGood ( const Stack S )
3.     {
4.     if ( S == NULL )
5.       {
6.         printf( “Stack routin recived bad stack \n “ );
7.       exit ( 1 );
8.       }
9.     }
```

# Thêm phần tử vào đầu Stack – push()

```
1.  void push ( int newEle, Stack S)
2.      {
3.      StInsistGood ( S ); // kiểm tra stack hợp lệ
4.      if( ++S→TopOfStack == S→MaxSize)
5.          { // nếu stack hiện thời đã đầy thì tăng gấp đôi kích thước.
6.          S→MaxSize *= 2;
7.          S→Array = realloc( S→Array, sizeof(StEtype) * S→MaxSize);
8.          if (S→Array == NULL)
9.              {
10.             printf( “can not extend the stack\n” );  exit (-1);
11.             }
12.         S→Array[ S→TopOfStack ] = newEle;
13.     }
```

# Stack – pop()

```
1. // ----------  Pop: lấy một phần tử ra khỏi stack  ----------
2. void pop( Stack S )
3.     {
4.     if ( StIsEmpty( S ) )
5.       printf( " Error: can not Pop an empty stack\n" );
6.     else
7.       S→TopOfStack --;
8.     }
```

# Stack –top()

```
1. // ---------- Top: lấy giá trị của phần tử trên đỉnh stack ---------
2. int top ( const Stack S )
3.     {
4.     if ( StIsEmpty ( S ) )
5.       printf ( “Error: can not Top an empty stack \n” );
6.     else
7.       return S→Array [ S→TopOfStack ];
8.     }
```

# Stack – StRecycle()

```
1. /*  Sau khi sử dụng stack, trước khi dừng chương trình, gọi
   StRecycle để giải phóng bộ nhớ. */
2. int StRecycle ( Stack S )
3.     {
4.     StInsistGood ( S );
5.     free ( S→Array );
6.     free ( S );
7.     }
```

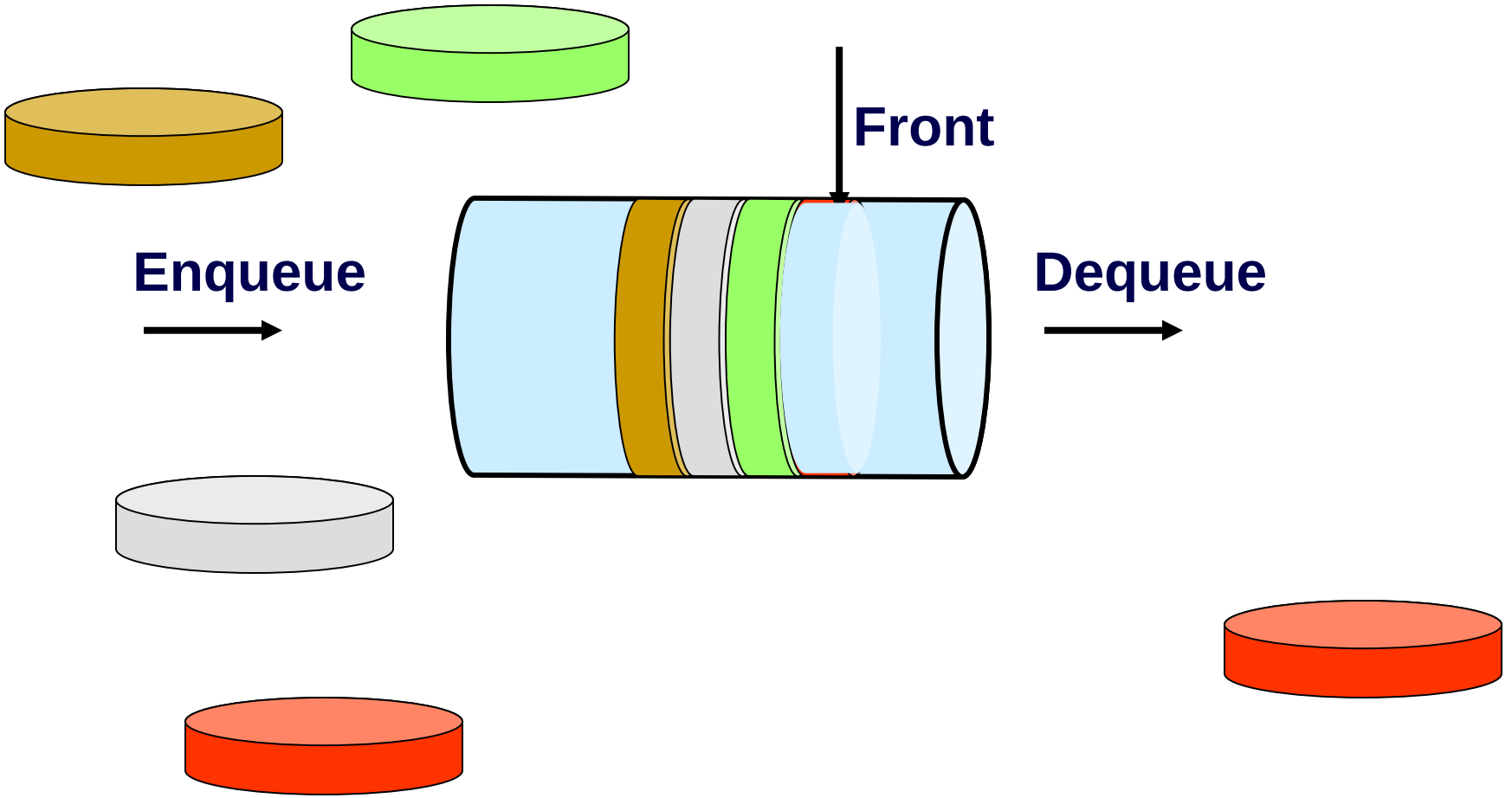
# Hàng đợi - Queue

- Queue (hàng đợi) là cấu trúc dữ liệu trong đó các truy xuất đến phần tử trong queue chỉ thực hiện được với phần tử chèn vào trước nhất.

- FIFO: First In First Out – Vào trước ra trước

- Các thao tác trong Queue:
  - Enqueue: chèn phần tử vào cuối hàng đợi.
  - Dequeue: lấy phần tử ra khỏi hàng đợi ờ đầu hàng đợi và lấy thông tin của phần tử này.
  - Front: kiểm tra phần tử đầu hàng đợi.

# Queue



# Khai báo kiểu Queue

```
1.  typedef struct QueueStr
2.      {
3.      int * Array; /* mảng các phần tử trong queue */

4.      int Front; /* chỉ số của phần tử đầu queue */
5.      int Back; /* chỉ số của phần tử cuối queue */
6.      int Size; /* số phần tử hiện tại */
7.      int MaxSize; /* Kích thước tối đa của Queue */
8.      } * Queue;

9.  static const QuInitSize = 5;   /* đầu tiên tạo queue có 5 phần tử */
```

# Kiểm tra Queue rỗng

```
1.  int QuIsEmpty ( const Queue Q )
2.      {
3.      QuInsisGood ( Q );
4.      return (Q→Size == 0);
5.      }

6.  // ---- KIểm tra xem có cần tăng kích thước Queue ----
7.  Int Increment ( int QParam, int Qsize)
8.      {
9.      if ( ++QParam == Qsize)
10.        return 0;
11.     else
12.        return QParam;
13.     }
```

# Queue - Khởi tạo

```
1.  Queue QuMakeEmpty ( Queue Q )
2.      {
3.      if ( Q == NULL )
4.          {
5.          if ( !( Q = malloc (sizeof( struct QueuStr))))
6.              return NULL;
7.          Q->Array = malloc (sizeof( QuEType ) * QuInitSize );
8.          if ( Q->Array == NULL)
9.              return NULL;
10.         Q->MaxSize = QuInitSize;
11.         }
12.     Q->Size = 0;
13.     Q->Front = 0;
14.     Q->Back = -1;
15.     return Q;
16.     }
```

# Lấy phần tử ra khỏi Queue

```
1.  // ---------- Dequeue ----------------
2.  void Dequeue (QuEType * X, Queue Q )
3.      {
4.      if ( QuIsEmpty (Q))
5.          printf (“Error: cannot Dequeue an empty queue\n”);
6.      else {
7.          *X = Q->Array [Q->Front ];
8.          Q->Front = Increment (Q->Front, Q->MaxSize);
9.          Q->Size--;
10.         }
11.     }
```

# Điều chỉnh Queue

```
1.  /* khi kích thước Queue được tăng lên gấp đôi, sự sắp xếp mảng có thể
    bị sai. Hàm FixWraparound giúp điều chỉnh lại và hạn chế số lần di
    chuyển các phần tử trong mảng */
2.  void FixWraparound (Queue Q)
3.      {
4.      const int OrigSz = Q->MaxSize / 2;
5.
6.      if (Q->Front < OrigSz / 2 )  {
7.          mempcy ( &Q->Array[OrigSz], &Q->Array[0],
                                  &Q->Front * sizeof(QuEType));
8.          Q->Back += QrigSz;
9.          }
10.     else  {
11.         memcpy (&Q->Array[OrigSz + Q->Front], &Q->Array[Q->Front],
                                  (OrigSz – Q->Front) * sizeof(QuEType));
12.         Q->Front += OrigSz;
13.         }
14.     }
```

# Đưa phần tử mới vào Queue

```
1.  void Enqueue (QuEType X, Queue Q)
2.      {
3.      QuInsistGood (Q);
4.      if (Q->Size == Q->MaxSize )   {
5.          Q->MaxSize *= 2;
6.          Q->Array = realloc (Q->Array, sizeof (QuEType) * Q->MaxSize);
7.          if (Q->Array == NULL)    {
8.              printf ( “Cannot extend the queue\n” );
9.              exit (-1);
10.             }
11.         if (Q->Front != 0)
12.             FixWraparound (Q);
13.     }
14.     Q->Back = Increment (Q->Back, Q->MaxSize);
15.     Q->Array[Q->Back] = X;
16.     Q->Size ++;
17.     }
```

# Lập chương trình cho máy tính

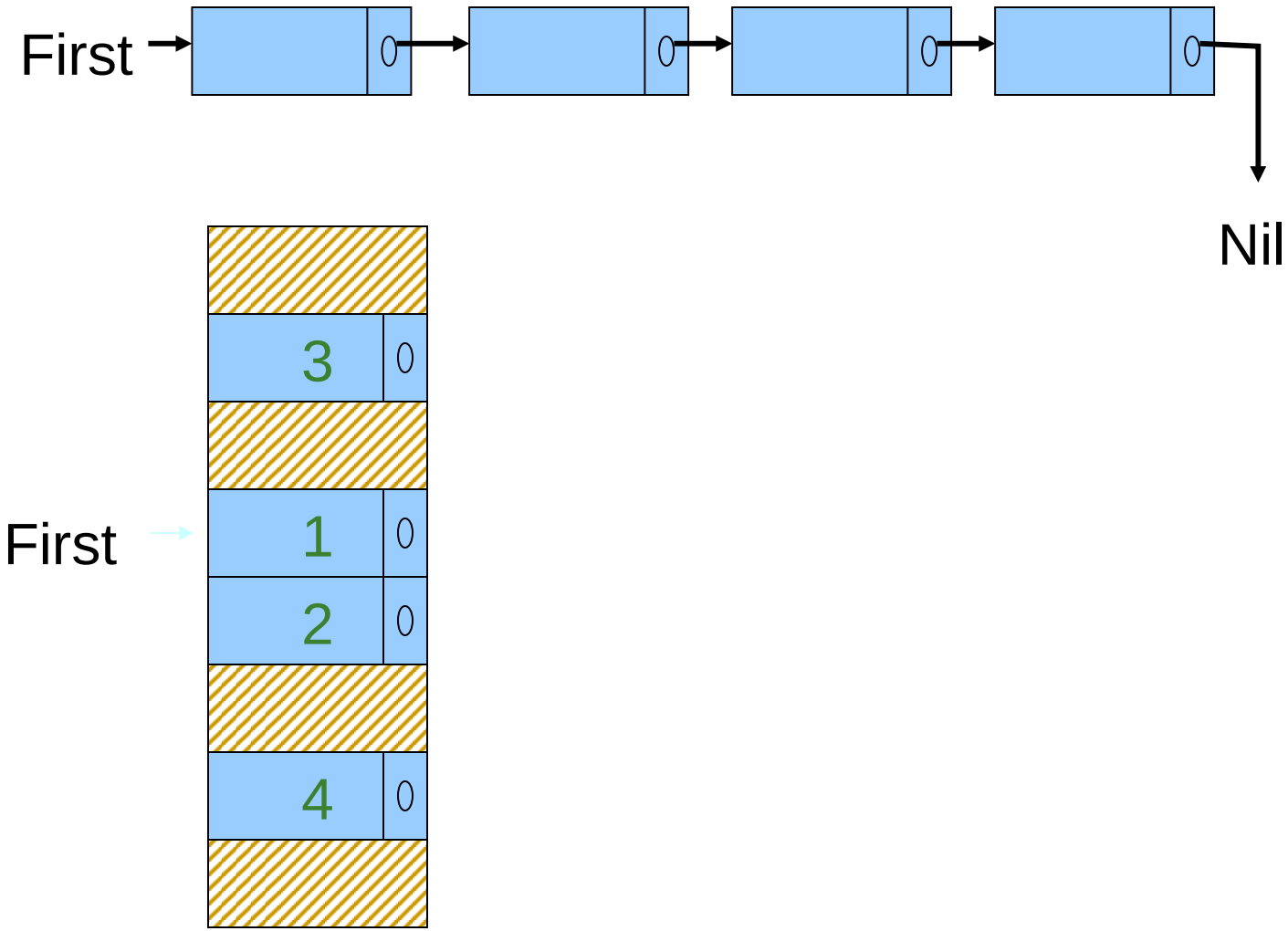
## CẤU TRÚC DỮ LIỆU – DANH SÁCH LIÊN KẾT (LINKED LISTS)

Lê Hà Thanh
Học kỳ 2, 2004-2005

# Khái niệm

- Khái niệm: (xem giáo trình Bài giảng kỹ thuật lập trình ...)
- Cấu trúc danh sách liên kết là cấu trúc động, việc cấp phát nút và giải phóng nút trên danh sách xảy ra khi chương trình đang chạy. Ta thường cấp phát nút cho danh sách liên kết bằng biến động.
- Các phần tử sẽ được cấp phát vùng nhớ trong quá trình thực thi chương trình, do đó chúng có thể nằm rải rác ở nhiều nơi khác nhau trong bộ nhớ (phân bố không liên tục).

# Biểu diễn trong bộ nhớ



# Liên Kết các nút trong danh sách

- Các phần tử trong danh sách liên kết kết nối với nhau theo dãy, trong đó:
  - First là con trỏ chỉ đến phần tử đầu tiên của danh sách liên kết.
  - Phần tử cuối của danh sách liên kết với vùng liên kết có nội dung NULL.
  - Mỗi nút của danh sách có trường info chứa nội dung của nút và trường next là con trỏ chỉ đến nút kế tiếp trong danh sách.

# Các đặc tính

- Cấu trúc DSLK là cấu trúc động, các nút được cấp phát hoặc giải phóng khi chương trình đang chạy.

- DSLK rất thích hợp khi thực hiện các phép toán trên danh sách thường bị biến động. Trong trường hợp xoá hay thêm phần tử trong DSLK thì ta không dời các phần tử đi như trong mảng mà chỉ việc hiệu chỉnh lại trường next tại các nút đang thao tác. Thời gian thực hiện các phép toán thêm vào và loại bỏ không phụ thuộc và số phần tử của DSLK.

# Hạn chế của Danh Sách Liên Kết

- Vì mỗi nút của DSLK phải chứa thêm trường next nên DSLK phải tốn thêm bộ nhớ.

- Tìm kiếm trên DSLK không nhanh vì ta chỉ được truy xuất tuần tự từ đầu danh sách.

# Khai báo và Khởi tạo

```
1. struct node
2.     {
3.     int info;
4.     struct node * next;
5.     };

6. typedef struct node *NODEPTR;
```

- Khai báo biến First quản lý DSLK:

```
NODEPTR First;
```

- Khởi tạo DSLK: First = NULL;

# Chú ý

- Thành phần chứa nội dung có thể gồm nhiều vùng với các kiểu dữ liệu khác nhau.

- Thành phần liên kết cũng có thể nhiều hơn 1 nếu là danh sách đa liên kết hoặc danh sách liên kết kép.

- First là con trỏ trỏ đến phần tử đầu tiên của danh sách liên kết, nó có thể là kiểu con trỏ, và cũng có thể là một struct có hai thành phần: First – con trỏ trỏ đến phần tử đầu tiên của DSLK, và Last – con trỏ trỏ đến phần tử cuối cùng của DSLK.

```
struct Linked_List {
        NODEPTR First;
        NODEPTR Last;
       }
```

# Các Phép Toán Trên DSLK - Tao Danh Sách

- Initialze: Khởi động một DSLK. Ban đầu DSLK chưa có phần tử.

```
void Initialize (NODEPTR *First)
      {
       * First = NULL;
      }
```

- New_Node(): cấp phát một nút cho DSLK. Hàm New_Node trả về địa chỉ của nút vừa được cấp phát.

```
NODEPTR New_Node ()
     {
       NODEPTR p;
       p = (NODEPTR) malloc (sizeof (struct node));
       return (p);
     }
```

# Tạo danh sách – Thêm vào đầu DS

- Insert_First(): thêm một nút có nội dung x vào đầu DSLK.

```
1. void Insert_First (NODEPTR *First, int x)
       {
2.       NODEPTR p;

3.       p = New_Node ();
4.       p->info = x;
5.       p->next = *First;
6.       *First = p;
7.     }
```

# Tạo danh sách – Thêm vào cuối DS

- Insert_After(): thêm một nút có nội dung x vào sau nút có địa chỉ p trong DSLK First.

```
1.  void Insert_After (NODEPTR p, int x)
2.      {
3.          NODEPTR q;
4.          if (p == NULL)
5.              printf (“Cannot insert new node!\n”);
6.          else
7.              {
8.              q = New_Node ();
9.              q->info = x;
10.              q->next = p->next;
11.             p->next = q;
12.            }
13.     }
```

# Cập Nhật Danh Sách

- Free_Node(): hủy một nút đã cấp phát và trả vùng nhớ về cho memory heap.

```
void Free_Node (NODEPTR p)
    {
       free (p);
    }
```

- Empty(): Kiểm tra danh sách rỗng.

```
int Empty (NODEPTR *First)
    {
       return (*First == NULL ? TRUE : FALSE);
    }
```

# Xóa phần tử đầu DS

- Delete_First(): Xoá phần tử đầu danh sách.

```
1.  void Delete_First (NODEPTR *First)
2.      {
3.          NODEPTR p;
4.          if (Empty (First))
5.              printf (“List is empty. No deletion performed!\n”);
6.          else    {
7.              p = *First;
8.              *First = p->next;

9.              Free_Node (p);
10.             }
11.     }
```

# Xóa phần tử cuối DS

- Delete_After(): Xoá phần tử đứng sau nút có địa chỉ p.

```
1.  void Delete_After (NODEPTR p)
2.      {
3.          NODEPTR q;
4.          if (p == NULL || p->next == NULL)
5.              printf (“Cannot delete!\n”);
6.          else    {
7.              q = p->next;     // q is the node that will be deleted
8.              p->next = q->next;

9.              Free_Node (q);
10.             }
11.     }
```

# Xóa toàn bộ Danh Sách

- Delete_All(): Xoá toàn bộ danh sách.
  Có thể gán First = NULL để xóa toàn bộ danh sách nhưng phần vùng nhớ đã cấp cho các phần tử trong DS không được giải phóng. Do đó chúng ta dùng giải thuật sau.

```
1.  void Delete_All (NODEPTR *First)
2.      {
3.          NODEPTR p;
4.          while (*First != NULL) // reach to end ?
5.              {
6.              p = *First;
7.              *First = (*First)->next;     // *First = p->next;
8.              Free_Node (p);
9.              }
10.     }
```

# Duyệt Danh Sách

- Traverse(): Duyệt qua toàn bộ danh sách (để liệt kê dữ liệu hoặc đếm số nút trong DS,...)

```
1.  void Traverse (NODEPTR *First)
2.      {  NODEPTR p;
3.         int count = 0;

4.         p = *First;
5.         if (p == NULL)
6.            printf (“List is empty!\n”);
7.         while (p != NULL)   {    // reach to end ?
8.            printf (“\n  %5d%8d”, count++, p->info);
9.            p = p->next;
10.           }
11.     }
```

# Tìm Kiếm trong Danh Sách

- Search(): Tìm nút đầu tiên trong DS có info bằng với x. Do đây là DSLK nên ta phải tìm từ đầu DS.
  Nếu tìm thấy nút có (info == x) thì trả về địa chỉ của nút, nếu không, trả về NULL.

```
1.  NODEPTR Search (NODEPTR *First, int x)
2.      {  NODEPTR p;

3.         p = *First;
4.         // not reach to end and not found
5.         while (p != NULL && p->info != x)
6.             p = p->next;

7.         return (p);
8.      }
```

# Sắp Xếp trong Danh Sách

- Selection_Sort(): sắp xếp DSLK theo thứ tự info tăng dần.

- Thuật toán:
  - So sánh tất cả các phần tử của DS để chọn ra một phần tử nhỏ nhất đưa về đầu DS;
  - sau đó, tiếp tục chọn phần tử nhỏ nhất trong các phần tử còn lại để đưa về phần tử thứ hai trong DS.
  - Quá trình lặp lại cho đến khi chọn được phần tử nhỏ nhất thứ (n-1)

# Sắp Xếp trong Danh Sách

```
1.  void Selection_Sort (NODEPTR *First)
2.      {  NODEPTR p, q, pmin;
3.         int min;
4.         for (p = *First; p->next != NULL; p = p->next)    {
5.            min = p->info;
6.            pmin = p;
7.            for (q = p->next; q != NULL; q = q->next)
8.               if (min > q->info)   {
9.                  min = q->info;
10.                 pmin = q;
11.              }
12.           // hoan doi truong info cua hai nut p va pmin
13.           pmin->info = p->info;
14.           p->info = min;
15.           }
16.     }
```