

NGÔN NGỮ LẬP TRÌNH 1-C

Thông tin về giáo viên

TT	Họ tên giáo viên	Học hàm	Học vị	Đơn vị công tác
1	Trần Cao Trường	GV	ThS	Bộ môn Khoa học máy tính
2	Hà Chí Trung	GVC	TS	Bộ môn Khoa học máy tính
3	Nguyễn Việt Hùng	GV	TS	Bộ môn Khoa học máy tính
4	Phan Thị Hải Hồng	GV	ThS	Bộ môn Khoa học máy tính
5	Nguyễn Trung Tín	TG	TS	Bộ môn Khoa học máy tính
6	Vi Bảo Ngọc	TG	ThS	Bộ môn Khoa học máy tính

- Thời gian, địa điểm làm việc: 7h-17h, Tầng 2, nhà A1
- Địa chỉ liên hệ: Bộ môn Khoa học máy tính, khoa Công nghệ thông tin
- Điện thoại, email: 0983836615, k12_khmt@gmail.com

Thông tin chung về học phần

- Tên học phần: **Ngôn ngữ lập trình 1-C**
- Mã học phần:
- Số tín chỉ:3
- Học phần (bắt buộc hay lựa chọn): bắt buộc
- Các học phần tiên quyết: lập trình cơ bản
- Các yêu cầu đối với học phần (nếu có):
- Giờ tín chỉ đối với các hoạt động:
 - Nghe giảng lý thuyết: 27
 - Làm bài tập trên lớp: 18
 - Thảo luận:
 - Thực hành, thực tập (ở PTN, nhà máy, thực tập...):
 - Hoạt động theo nhóm:
 - Tự học: 90
- Khoa/Bộ môn phụ trách học phần, địa chỉ: Bộ môn Khoa học máy tính, khoa Công nghệ thông tin

Mục tiêu của học phần

- Kiến thức: Giới thiệu cho học viên những kiến thức cơ bản của ngôn ngữ C
- Kỹ năng: Kết thúc môn học viên có thể xây dựng được các chương trình cơ bản với ngôn ngữ C
- Thái độ, chuyên cần: Tạo cho học viên tác phong làm việc nhóm; có khả năng tiếp cận, nghiên cứu và sử dụng các ngôn ngữ lập trình

Nội dung chi tiết học phần

Chương	Nội dung	Số tiết
1	Tổng quan về ngôn ngữ C	1
2	Các yếu tố cơ bản của ngôn ngữ C	5
3	Mảng và con trỏ	6
4	Chương trình con- hàm	9
5	Dữ liệu kiểu cấu trúc	3
6	Danh sách liên kết	6
7	Ngăn xếp	6
8	Hàng đợi	3
9	File và các thao tác I/O	6

Tài liệu tham khảo

- Giáo trình kỹ thuật lập trình C cơ bản và nâng cao, Phạm Văn Ất, Nhà xuất bản KHKT, 2009
- Bài tập ngôn ngữ C từ A đến Z; Huỳnh Tấn Dũng, Hoàng Đức Hải; Nhà xuất bản lao động, xã hội; 2005
- Brian Kernighan, Dennis Ritchie: *The C Programming Language*. Also known as K&R — The original book on C 2nd, Prentice Hall 1988; ISBN 0-13-110362-8. ANSI C.
- Robert Sedgewick: *Algorithms in C*, Addison-Wesley, ISBN 0-201-31452-5 (Part 1–4) and ISBN 0-201-31663-3 (Part 5)

Tổng quan về ngôn ngữ C

Tổng quan về ngôn ngữ C

- Mục đích, yêu cầu
 - Giới thiệu cơ bản ngôn ngữ C
 - Sinh viên cần nắm được cấu trúc cơ bản ngôn ngữ C, biết cài đặt và sử dụng các trình biên dịch C
- Hình thức tổ chức dạy học: lý thuyết, thảo luận, bài tập, thực hành, tự học
- Thời gian: tiết 1- tuần 1
- Địa điểm: theo phân công P2

Nội dung

- Giới thiệu về ngôn ngữ C
- Giới thiệu môi trường biên dịch C
- Cấu trúc cơ bản chương trình C

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. }
```

12	c
7	b
5	a

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	<i>identifiers constants</i>	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	$X < Y$	$X \leq Y$	$X > Y$	$X \geq Y$	$X \neq Y$	$X == Y$	$X=Y$	$X<<Y$	$X>>Y$
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).
- $i = i + 1;$ $i++;$ $++i;$
- $i = i - 1;$ $i--;$ $--i;$
- $i = i + 3;$ $i += 3;$
- $i = i * j;$ $i *= j;$

Các yếu tố cơ bản của ngôn ngữ

C

Các yếu tố cơ bản của ngôn ngữ C

- Mục đích, yêu cầu
 - Giới thiệu các yếu tố cơ bản của ngôn ngữ C
 - Sinh viên cần nắm được cách sử dụng biến, hằng, các cấu trúc điều khiển
- Hình thức tổ chức dạy học: lý thuyết: 2, thảo luận, bài tập: 3, thực hành, tự học: 10
- Thời gian: 6 tiết - tuần 11 và tuần 12
- Địa điểm: theo phân công P2

Nội dung

- Từ vựng dùng trong C
- Biểu thức
- Vào ra dữ liệu chuẩn
- Các câu lệnh điều khiển chương trình

Khái Niệm Cơ Bản

- Một **biểu thức** là bất kỳ sự tính toán nào mà cho ra một giá trị.
- Một biểu thức **ước lượng** một giá trị nào đó.

Toán Tử Toán Học & Luận Lý

Toán tử	Tên	Ví dụ
+	Cộng	$12 + 4.9$ // cho 16.9
-	Trừ	$3.98 - 4$ // cho -0.02
*	Nhân	$2 * 3.4$ // cho 6.8
/	Chia	$9 / 2.0$ // cho 4.5
%	Lấy phần dư	$13 \% 3$ // cho 1

Toán tử	Tên	Ví dụ
==	So sánh bằng	$5 == 5$ // cho 1
!=	So sánh không bằng	$5 != 5$ // cho 0
<	So sánh nhỏ hơn	$5 < 5.5$ // cho 1
<=	So sánh nhỏ hơn hoặc bằng	$5 <= 5$ // cho 1
>	So sánh lớn hơn	$5 > 5.5$ // cho 0
>=	So sánh lớn hơn hoặc bằng	$6.3 >= 5$ // cho 1

Toán Tử Luận Lý & Trên Bit

Toán tử	Tên	Ví dụ
!	Phủ định luận lý	!(5 == 5) // được 0
&&	Và luận lý	5 < 6 && 6 < 6 // được 0
	Hoặc luận lý	5 < 6 6 < 5 // được 1

0: SAI (false)

Khác 0: ĐÚNG (true)

Toán tử	Tên	Ví dụ
~	Phủ Định Bit	~'\011' // được '\366'
&	Và bit	'\011' & '\027' // được '\001'
	Hoặc bit	'\011' '\027' // được '\037'
^	Hoặc exclusive bit	'\011' ^ '\027' // được '\036'
<<	Dịch trái bit	'\011' << 2 // được '\044'
>>	Dịch phải bit	'\011' >> 2 // được '\002'

Toán Tử Tăng/Giảm & Khởi Tạo

Toán Tử	Tên	Ví dụ
++	Tăng một (tiền tố)	++k + 10 // được 16
++	Tăng một (hậu tố)	k++ + 10 // được 15
--	Giảm một (tiền tố)	--k + 10 // được 14
--	Giảm một (hậu tố)	k-- + 10 // được 15

Toán Tử	Ví dụ	Tương đương với
=	n = 25	
+=	n += 25	n = n + 25
-=	n -= 25	n = n - 25
*=	n *= 25	n = n * 25
/=	n /= 25	n = n / 25
%=	n %= 25	n = n % 25
<<=	n <<= 4	n = n << 4
>>=	n >>= 4	n = n >> 4

Toán Tử Điều Kiện, Phẩy & Lấy Kích Thước

Toán tử điều kiện

```
min = (m < n ? m++ : n++);
```

Toán tử phẩy

```
min = (m < n ? mCount++, m : nCount++, n);
```

Toán tử lấy kích thước

```
cout << "float size = " << sizeof(float) << " bytes\n";
```

Độ Ưu Tiên Của Các Toán Tử

Mức	Toán tử							Loại	Thứ tự
Cao nhất	::							Một ngôi	Cả hai
	()	[]	->	.				Hai ngôi	Trái tới phải
	+ -	++ --	! ~	* &	new delete	sizeof ()		Một ngôi	Phải tới trái
	->*	.*						Hai ngôi	Trái tới phải
	*	/	%					Hai ngôi	Trái tới phải
	+	-						Hai ngôi	Trái tới phải
	<<	>>						Hai ngôi	Trái tới phải
	<	<=	>	>=				Hai ngôi	Trái tới phải
	==	!=						Hai ngôi	Trái tới phải
	&							Hai ngôi	Trái tới phải
	^							Hai ngôi	Trái tới phải
								Hai ngôi	Trái tới phải
	&&							Hai ngôi	Trái tới phải
								Hai ngôi	Trái tới phải
	? :							Ba ngôi	Trái tới phải
	=	+= -=	*= /=	^= %=	&= =	<<= >>=		Hai ngôi	Phải tới trái
Thấp nhất	,	++,--						Hai ngôi	Trái tới phải

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. }
```

Câu lệnh điều kiện if

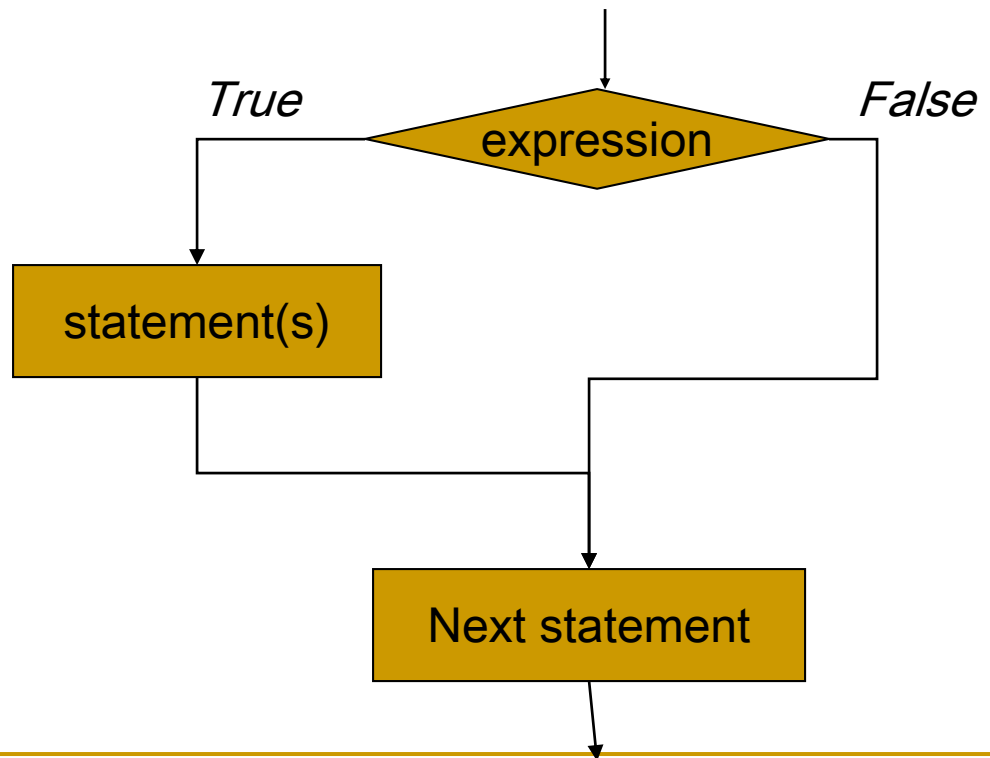
if (<dieu kien>)

{

/* 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 */

}

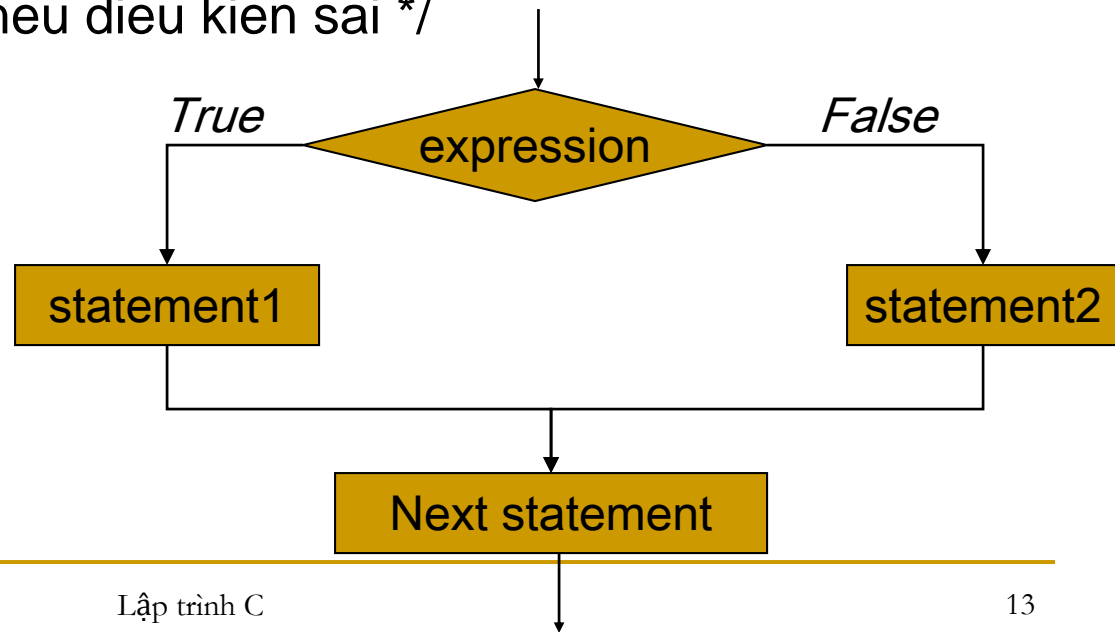
else

{

/* 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);
```

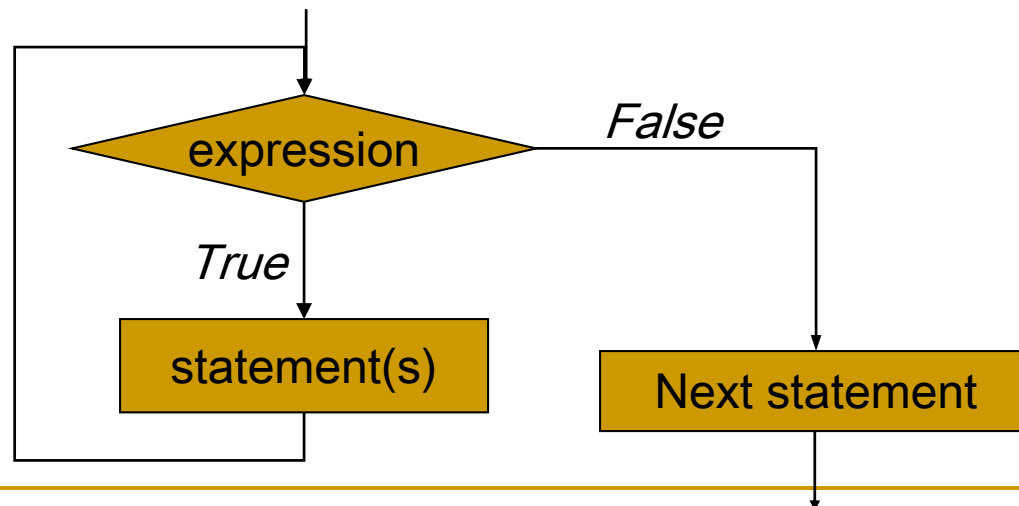
- Tuy nhiên, cần chú ý đến thứ tự các cặp lệnh if ... else ... khi lồng các lệnh if. Nếu không sẽ phát sinh lỗi.

```
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 (biểu thức điều kiện)
{các lệnh}
- 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

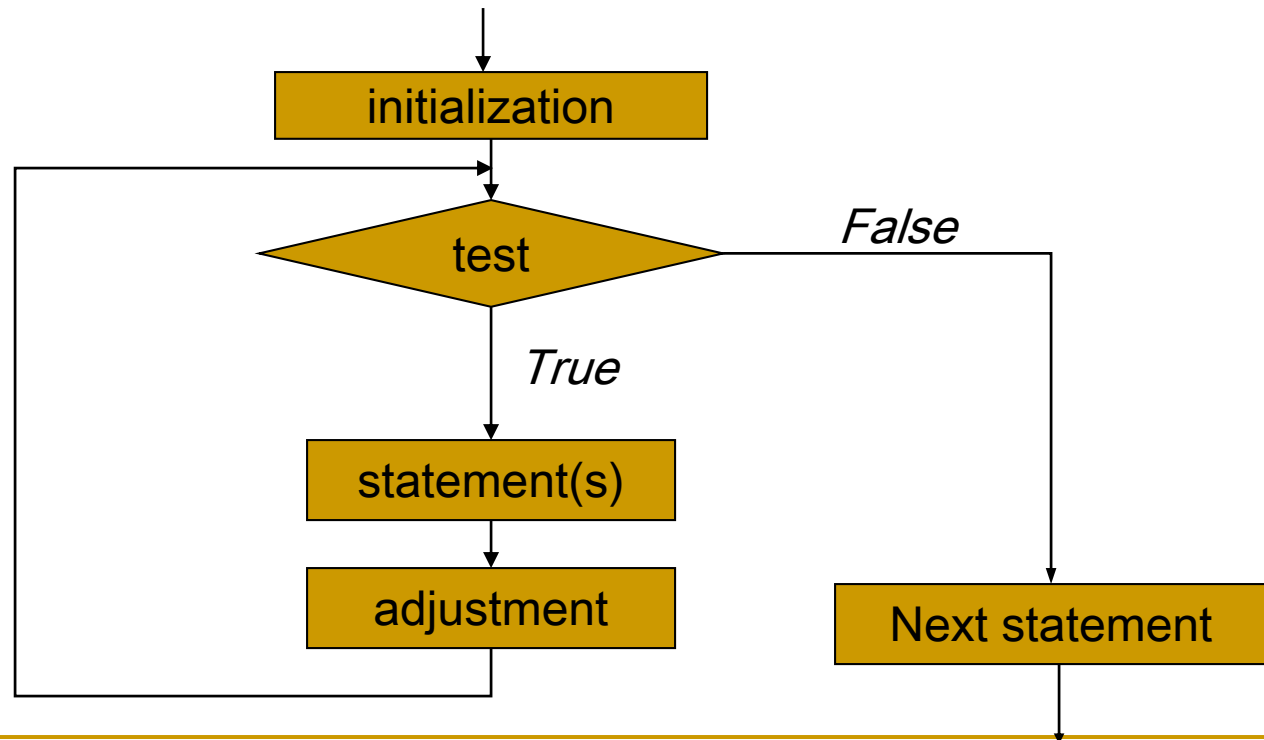
- In bảng đổi nhiệt độ từ độ Fahrenheit (oF) sang độ Celcius (oC).

```
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>

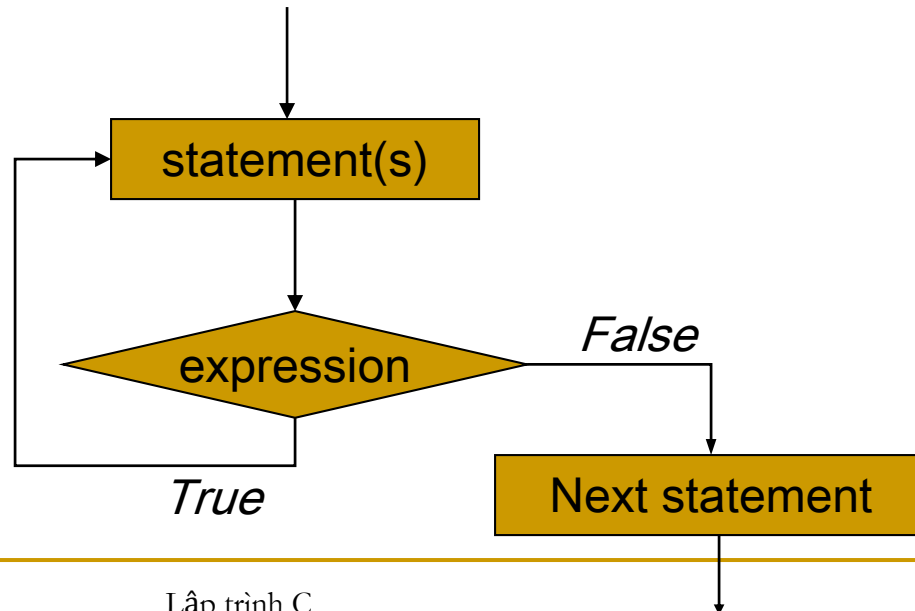
2. int main() {
3.     float a = 0;
4.     int i;
5.     for(i=0; i<=100; i+=10) {
6.         printf("%6.2f degrees F = %6.2f degrees C\n",
7.               a, (a - 32.0) * 5.0 / 9.0);
8.         a = a + 10;
9.     }
10.    return 0;
11. }
```

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("  Chương trình giai phương trình bậc thap \n");
12.         printf("  1. Giai phương trình bậc 1:  $ax + b = 0$  \n");
13.         printf("  2. Giai phương trình bậc 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

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_ending 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

- `#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
```

Mảng và con trỏ

Mạng và con trỏ

- Mục đích, yêu cầu
 - Giới thiệu về cách sử dụng mạng, con trỏ và mối liên hệ của chúng
 - Sinh viên cần nắm được cách sử dụng mạng, con trỏ và mối liên hệ của chúng
- Hình thức tổ chức dạy học: lý thuyết: 3, thảo luận, bài tập: 3, thực hành, tự học: 12
- Thời gian: 6 tiết - tuần 3 và tuần 4
- Địa điểm: theo phân công P2

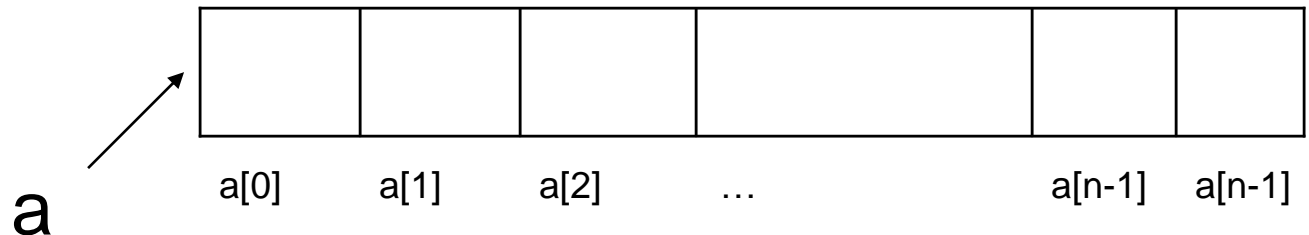
Nội dung

- Mạng
- Con trỏ
- Liên hệ giữa con trỏ và mảng
- Cấp phát bộ nhớ động

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 $i \geq \text{array_size}$ 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. // nhập giá trị cho các phần tử mảng
 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.     }

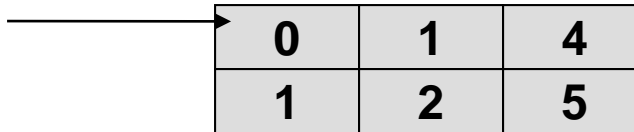
9. void getArray(int *a, int size)
10.    {for(int i=0; i<size; i++) {
11.        printf("a[%d]=");
12.        scanf("%d", &a[i]);
13.    }
14. }
```

Mảng nhiều chiều

- Khai báo mảng 2 chiều:

```
type name[row_size][column_size];
```

SumSquares



0	1	4
1	2	5

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

StudentScores

Student1

Student2

Student3

Student4

Student5

0	1	4	?	?	?
1	2	5	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?

0	1	4	?	?	?	1	2	5	?	?	?
---	---	---	---	---	---	---	---	---	---	---	---

Student1

Student2

Kiểu dữ liệu Con trỏ

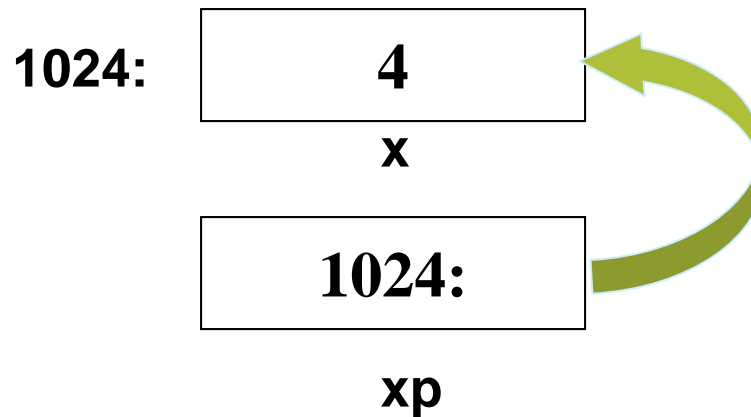
- Một biến kiểu con trỏ (pointer) chứa một tham chiếu (reference) đến một biến loại khác. Nói khác đi, biến con trỏ chứa địa chỉ ô nhớ của một biến.

```
int x;
```

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

```
x = 4;
```

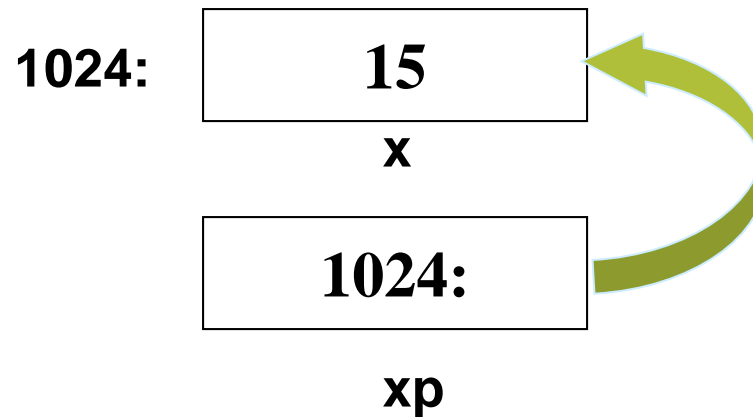
```
xp = &x;
```



Sử dụng Con trỏ

- truy cập vùng nhớ được chỉ bởi một con trỏ

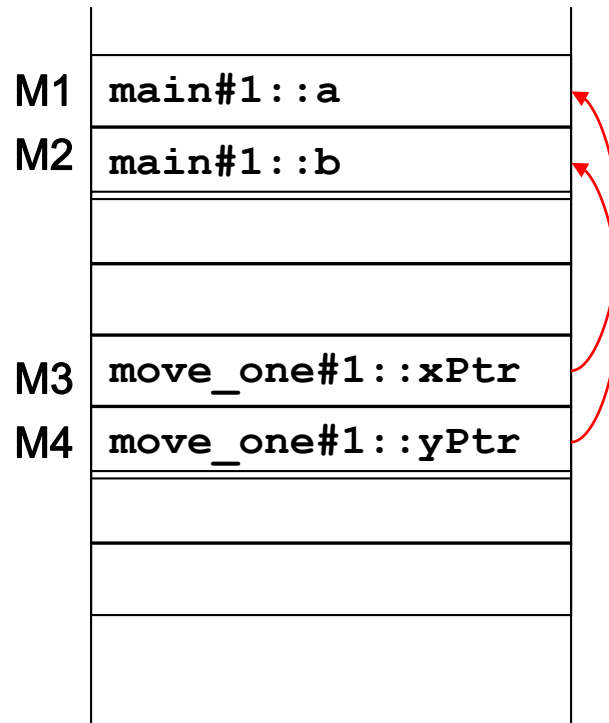
```
xp = &x;    /* gán địa chỉ x vào xp */  
*xp = 15;   /* gán giá trị 15 vào biến x */  
*xp = *xp + 1; /* cộng 1 vào x */
```



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);
}
```


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. }
```

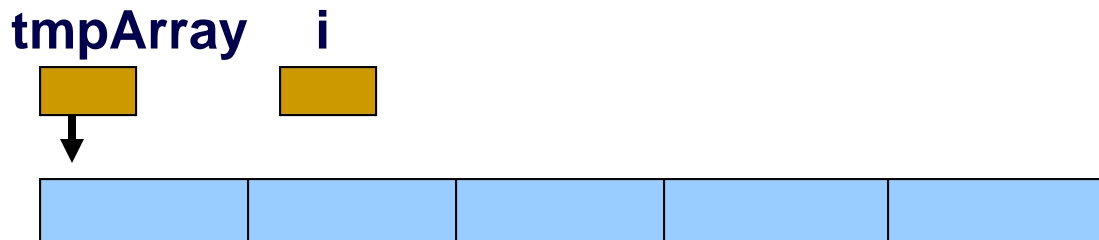
Giải phóng vùng nhớ

- 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);

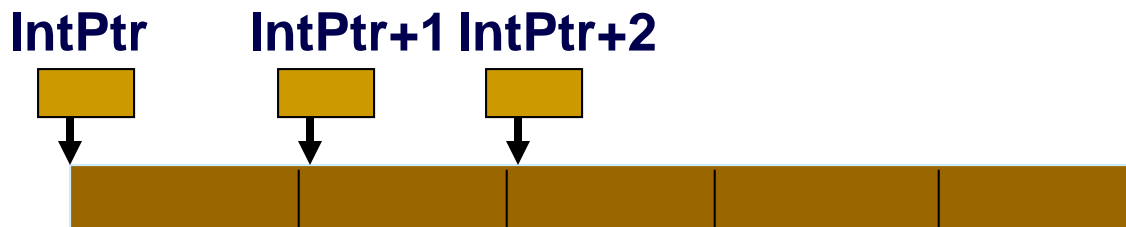
16. return 0;
17. }
```

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

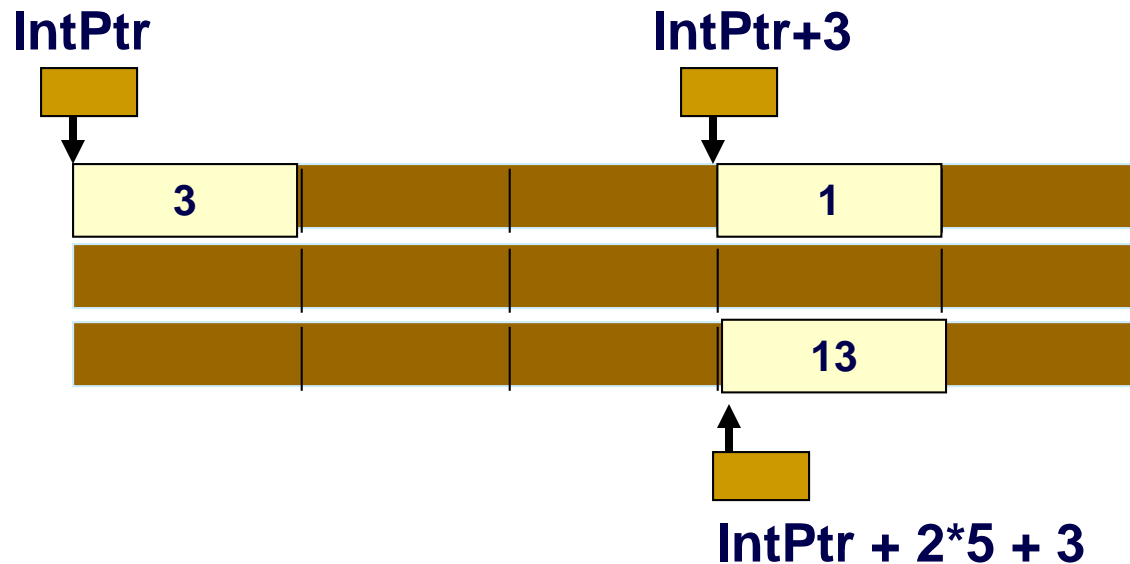
```
#define ROWS    5
#define COLS    3
int IntArr[ROWS][COLS] = {{3,4,2,1,0},
                           {5,6,7,8,9},
                           {10,11,12,13,14}};

int *IntPtr = IntArr;
```

- Tham chiếu đến phần tử trong mảng 2 chiều:

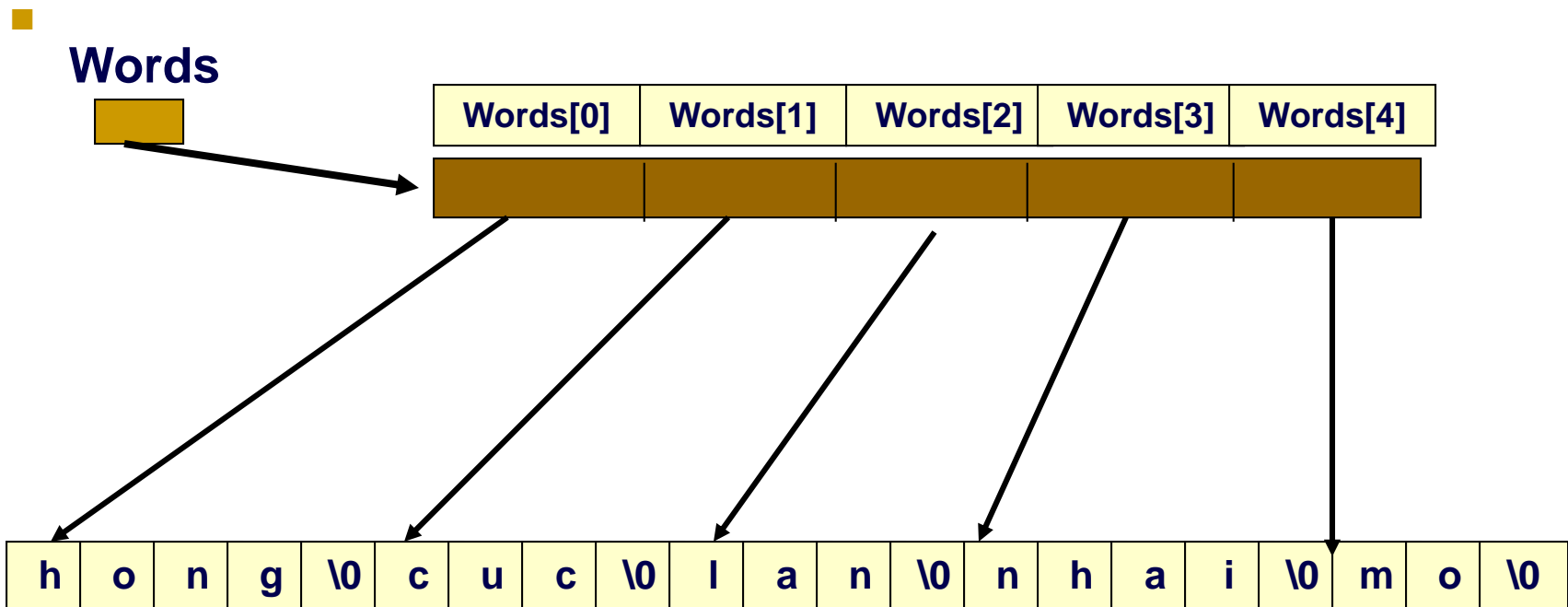
```
int val_at_00 = * IntPtr;// = IntArr[0][0]
int val_at_30 = * (IntPtr+3);// = IntArr[3][0]
int val_at_32 = * (IntPtr+3*5+2);// = IntArr[3][2]
int val_at_32 = * ( * (IntPtr+3)
```

Phép toán với con trỏ trong mảng



Mảng các chuỗi: char * []

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



Chương trình con-hàm

Chương trình con-hàm

- Mục đích, yêu cầu
 - Giới thiệu cách tạo, sử dụng hàm, truyền tham số cho hàm; con trỏ hàm; hàm đệ qui, hàm mẫu
 - Sinh viên cần nắm được cách cách tạo, sử dụng hàm, truyền tham số cho hàm; con trỏ hàm; hàm đệ qui, hàm mẫu
- Hình thức tổ chức dạy học: lý thuyết: 6, thảo luận, bài tập: 3, thực hành, tự học: 18
- Thời gian: 9 tiết - tuần 5, tuần 6 và tuần 7
- Địa điểm: theo phân công P2

Nội dung

- Giới thiệu
- Khai báo prototype
- Xây dựng hàm
- Tham số trong lời gọi hàm
- Con trỏ hàm
- Hàm đệ qui
- Hàm mẫu(**template function**)


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

M1	<code>main#1::Left = 5</code>
M2	<code>main#1::Right = 7</code>
M3	<code>Swap#1::Temp = ?</code>
M4	<code>Swap#1::x = 5</code>
M5	<code>Swap#1::y = 7</code>

M1	<code>main#1::Left = 5</code>
M2	<code>main#1::Right = 7</code>
M3	<code>Swap#1::Temp = 5</code>
M4	<code>Swap#1::x = 7</code>
M5	<code>Swap#1::y = 5</code>

Truyền bằng tham chiếu

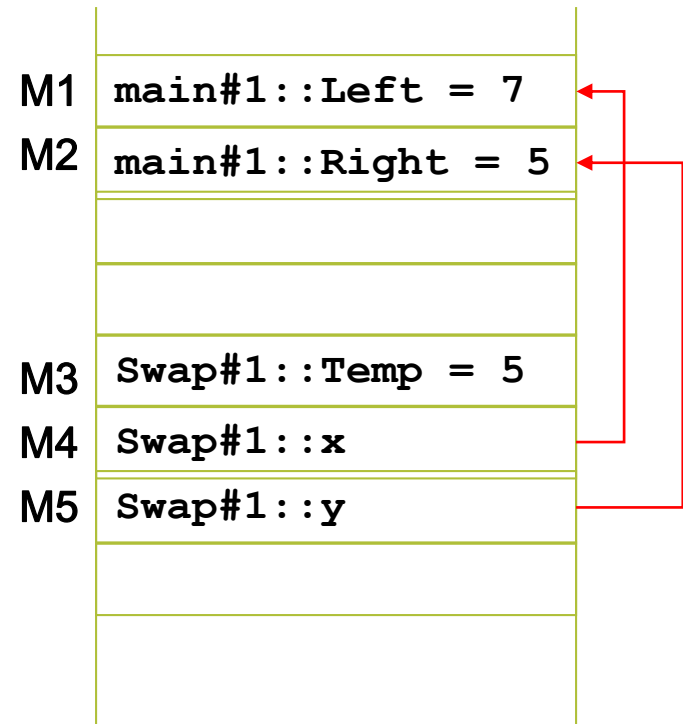
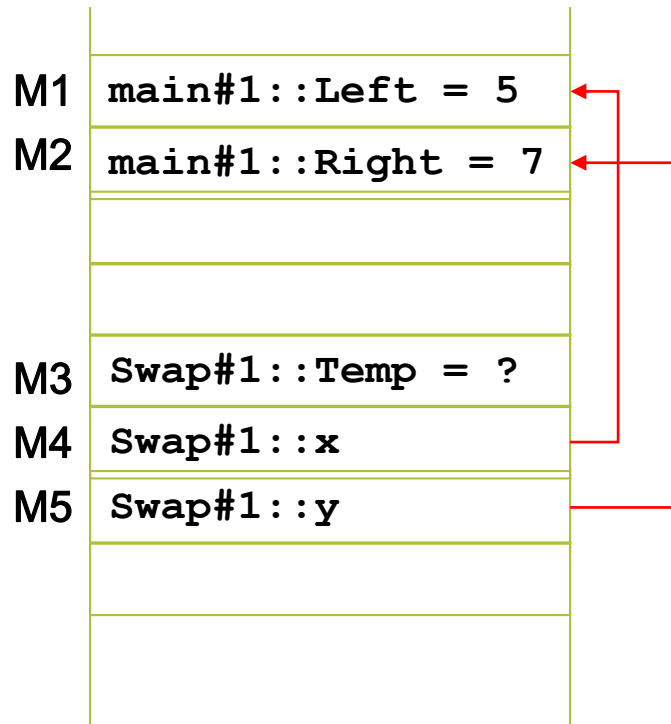
- `/* Swapping routine that does work */`
- `#include <stdio.h>`
- `void Swap(int &x, int &y)`
`{`
 `int Temp;`

 `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.

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>

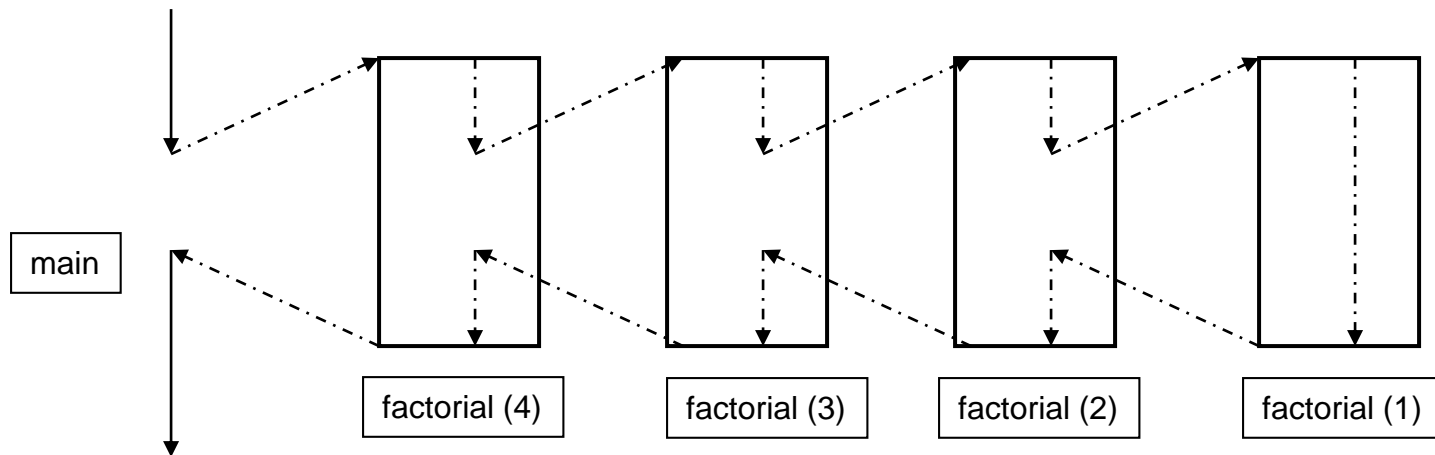
2. unsigned long int factorial(int n)
3. { if(n==0)
4.     return 1;
5.     return (n* factorial(n-1));
6. }

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

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

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

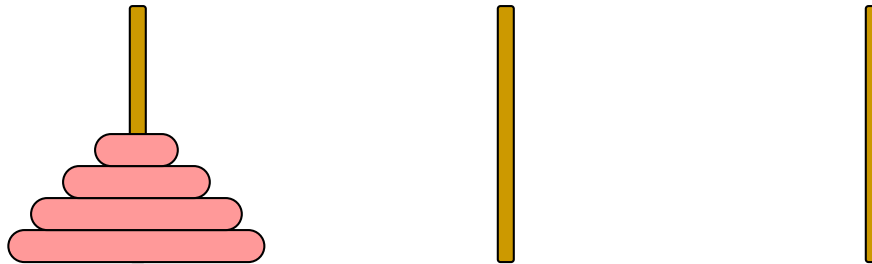
- Bài toán giải bằng thuật giải đệ quy phải có điều kiện dừng.
- Thuật toán đệ quy trên máy tính có thể bị giới hạn bởi dung lượng bộ nhớ do lời gọi hàm liên tiếp.



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

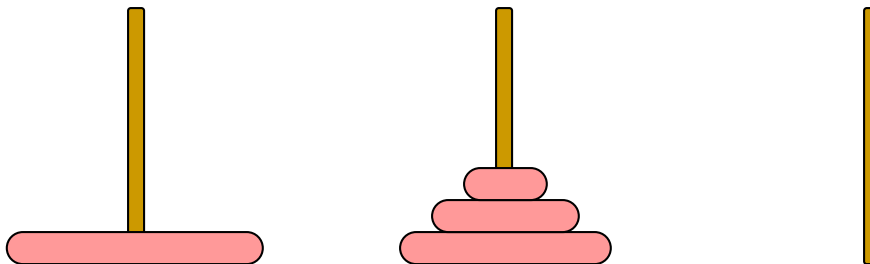
Bài toán Tháp Hà Nội

- 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.



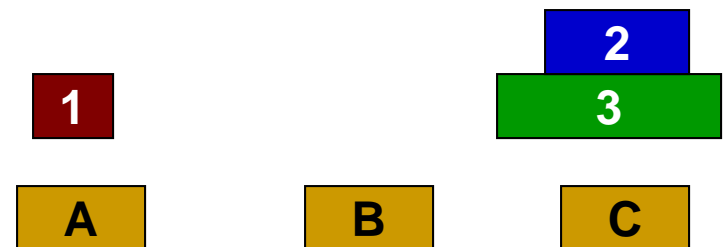
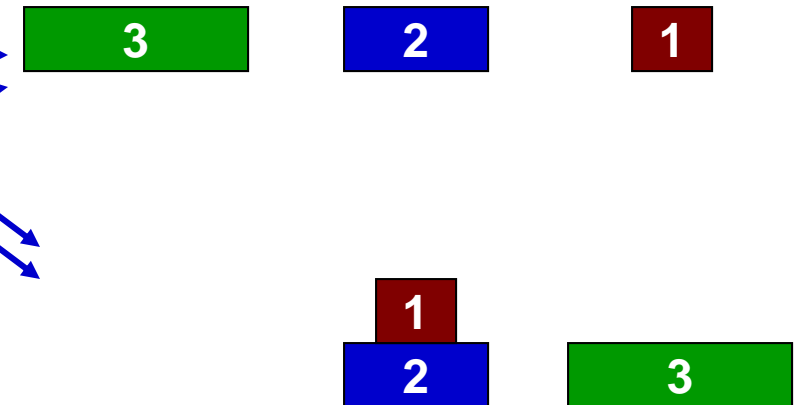
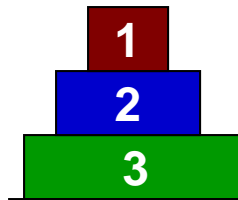
Thuật giải

- 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.



Tháp Hà Nội...

```
BC.EXE
Chuyen dia 1 tu cot A sang cot C
Chuyen dia 2 tu cot A sang cot B
Chuyen dia 1 tu cot C sang cot B
Chuyen dia 3 tu cot A sang cot C
Chuyen dia 1 tu cot B sang cot A
Chuyen dia 2 tu cot B sang cot C
Chuyen dia 1 tu cot A sang cot C
```



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. }
```

KHÁI QUÁT VỀ KHỬ ĐỆ QUY

- Là quá trình chuyển đổi 1 giải thuật đệ quy thành giải thuật không đệ quy.
- Chưa có giải pháp cho việc chuyển đổi này một cách tổng quát.
- Cách tiếp cận:
 - Khử đệ quy bằng vòng lặp
 - Khử đệ quy bằng Stack

KHỦ ĐỆ QUI BẰNG VÒNG LẶP

- Đi từ điều kiện biên đi tới điều kiện kết thúc.
- Ý tưởng: Lưu lại các trị của các lần tính toán trước làm dữ liệu cho việc tính toán của lần sau.

HÀM TÍNH N!

```
long GiaiThua( int n)
{ if (n<2) return 1;
  return n * GiaiThua(n-1);
}
```

Trị cần lưu

Điều kiện biên

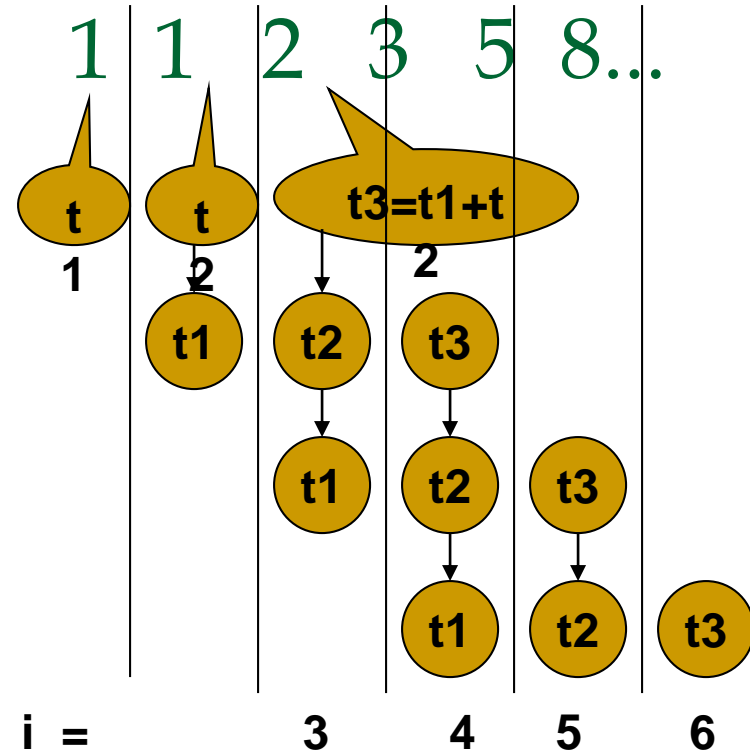
```
long GiaiThua( int n)
{ long K=1;
  for (int i =2; i<=n;i++) K=K*i;
  return K;
}
```

**K chính là kết quả của trị
giai thừa trước đó**

DÃY FIBONAXI

```
long Fibo(int n)
{ if (n<=2) return 1; // hai chặn
  return Fibo(n-2) + Fibo (n-1);
}
```

```
long Fibo(int n)
{ if (n<=2) return 1; // hai chặn
  long t1=1, t2=1;
  for (int i=3; i<=n;i++)
  { t3=t1+t2;
    t1=t2;
    t2= t3;
  }
  return t3;
}
```



Bài tập

1. Viết chương trình xuất n trị đầu tiên của 1 cấp số cộng có số hạng đầu là a (nhập từ bàn phím), công sai r (nhập từ bàn phím). Sử dụng kỹ thuật đệ quy để xây dựng hàm tính trị thứ i của 1 cấp số cộng này.
2. Dùng kỹ thuật đệ quy để giải phương trình $f(x)$ trong khoảng $[a,b]$ với sai số epsilon.
3. Viết chương trình nhập 1 mảng số int, nhập 1 trị x, tìm vị trí có x cuối cùng trong mảng. Dùng kỹ thuật đệ quy để tìm vị trí này.
4. Viết chương trình nhập 1 ma trận vuông các số int , nhập 1 trị x. Tìm vị trí <dòng,cột> có x dùng kỹ thuật đệ quy.

BÀI TẬP

5. Cài đặt thuật toán tìm UCLN bằng giải thuật đệ qui và không đệ qui
6. Cài đặt giải thuật chuyển từ số thập phân sang số nhị phân bằng đệ qui và không đệ qui
7. Cài đặt thuật toán tìm kiếm nhị phân bằng giải thuật đệ qui và không đệ qui
8. Cài đặt thuật toán sắp xếp QuickSort bằng giải thuật đệ qui và không đệ qui
9. Cài đặt thuật toán tìm nghiệm của phương trình trong khoảng $[a,b]$ bằng giải thuật đệ qui và không đệ qui

Dữ liệu kiểu cấu trúc

Dữ liệu kiểu cấu trúc

- Mục đích, yêu cầu
 - Giới thiệu khái niệm cấu trúc, cách khai báo, sử dụng cấu trúc
 - Sinh viên cần nắm được khái niệm cấu trúc, cách khai báo, sử dụng cấu trúc
- Hình thức tổ chức dạy học: lý thuyết: 3, thảo luận, bài tập:, thực hành, tự học: 6
- Thời gian: 3 tiết - tuần 8
- Địa điểm: theo phân công P2

Nội dung

- Khái niệm
- Cấu trúc tự trở
- Kiểu hợp

Cấu trúc (structure)

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

```
struct TenCauTruc
{
    << Kiểu DL >> << Trường 1 >> ;
    << Kiểu DL >> << Trường 2 >> ;
    << Kiểu DL >> << Trường 3 >> ;
    << ... >>
}
```

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 báo 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,
7.                  c01vta1[i].noisinh, c01vta1[i].maso);
8.          printf("\nTiep tục? (C/K)");
9.          tieptuc = (toupper(getch()) == 'C' ? 1 : 0);
10.         i++;
11.     }
```

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;
```

Danh sách liên kết

Danh sách liên kết

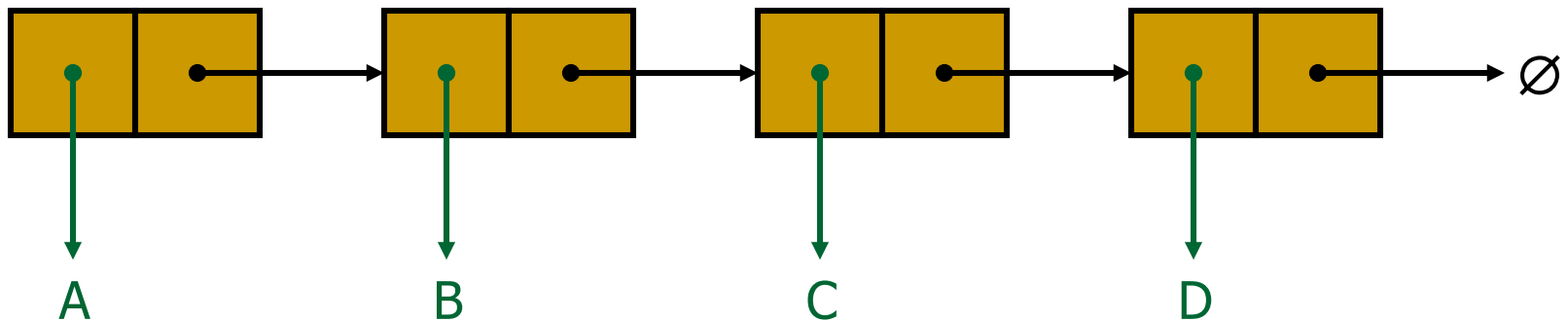
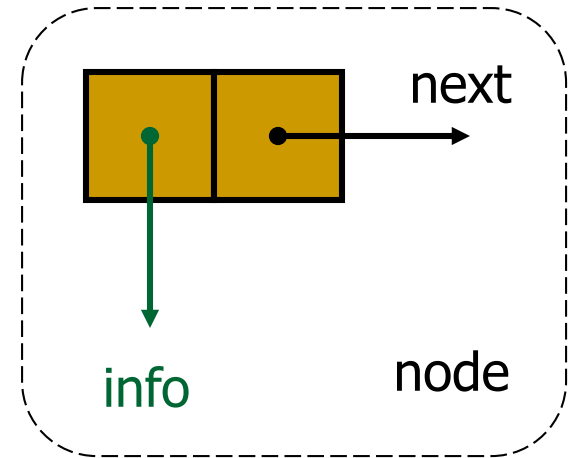
- Mục đích, yêu cầu
 - Giới thiệu khái niệm, ưu và nhược điểm của DSLK, cách cài đặt DSLK bằng mảng và con trỏ
 - Sinh viên cần nắm khái niệm, ưu và nhược điểm của DSLK, cách cài đặt DSLK bằng mảng và con trỏ
- Hình thức tổ chức dạy học: lý thuyết: 3, thảo luận, bài tập: 3, thực hành, tự học: 6
- Thời gian: 6 tiết - tuần 9 và tuần 10
- Địa điểm: theo phân công P2

Nội dung

- Khái niệm DSLK
- Ưu điểm và hạn chế của DSLK
- Các phép toán cơ bản của DSLK
- Bài tập

KHÁI NIỆM(1/2)

- Danh sách liên kết là một cấu trúc dữ liệu bao gồm một tập các nút, mà mỗi nút bao gồm:
 - Dữ liệu cần lưu trữ
 - Liên kết đến nút tiếp theo



KHÁI NIỆM(2/2)

- 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.
- Minh họa

KHAI BÁO

- **struct** node
 {
 int info;
 struct node* next;
 };
- **typedef struct** node* NODEPTR;

NỘI DUNG

- Khái niệm
- Ưu điểm và hạn chế
- Các phép toán cơ bản
- Bài tập

NỘI DUNG

- Khái niệm
- Ưu điểm và hạn chế
- Các phép toán cơ bản
- Bài tập

ƯU ĐIỂM

- Cấu trúc DSLK là cấu trúc động, các nút được cấp phát hoặc giải phóng khi chương trình đang chạy->kích thước của danh sách không phải khai báo trước
- DSLK rất thích hợp khi thực hiện các phép toán trên danh sách thường bị biến động như thêm hay xóa mà không phụ thuộc vào số phần tử của DS

HẠN CHẾ

- 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.

NỘI DUNG

- Khái niệm
- Ưu điểm và hạn chế
- Các phép toán cơ bản
- Bài tập

NỘI DUNG

- Khái niệm
- Ưu điểm và hạn chế
- Các phép toán cơ bản
- Bài tập

KHỞI TẠO DSLK

- Initialize: Khởi tạo một DSLK. Ban đầu DSLK chưa có phần tử.
- Cài đặt

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

CẤP PHÁT BỘ NHỚ CHO 1 NÚT

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

- Cài đặt

```
NODEPTR New_Node ()
```

```
{
```

```
    NODEPTR p;
```

```
    p=(NODEPTR)malloc(sizeof (struct node));
```

```
    return (p);
```

```
}
```

THÊM VÀO ĐẦU DANH SÁCH

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

```
void Insert_First (NODEPTR &First, int x)
{
    NODEPTR p;
    p = New_Node ();
    p->info = x;
    p->next = First;
    First = p;
}
```

THÊM VÀO GIỮA DANH SÁCH

- Insert_After(): thêm một nút có nội dung x vào sau nút có địa chỉ p trong ĐSLK First.
- Minh họa
- Cài đặt

```
void Insert_After (NODEPTR p, int x)
{
    NODEPTR q;
    if (p == NULL)
        printf ("Cannot insert new node!\n");
    else
    {
        q = New_Node ();
        q->info = x;
        q->next = p->next;
        p->next = q;
    }
}
```

THÊM VÀO CUỐI DANH SÁCH

- Insert_End(): thêm một nút có nội dung x vào cuối DSLK.
- Minh họa
- Cài đặt

```
void InsertEnd(NodePtr &First,int x)
{
    NodePtr r=NewNode();
    r->info=x;
    r->next=NULL;
    if(First!=NULL)
    {
        NodePtr temp=First;
        while(temp->next!=NULL)
            temp=temp->next;
        temp->next=r;
    }
    else
        First=r;
}
```


HỦY MỘT NÚT

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

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

KIỂM TRA DANH SÁCH RỖNG

- Empty(): Kiểm tra danh sách rỗng.
- Cài đặt:

```
int Empty (NODEPTR First)
{
    return (First == NULL ? 1 : 0);
}
```

XÓA NÚT ĐẦU DANH SÁCH

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

```
void Delete_First (NODEPTR &First)
{
    NODEPTR p;
    if (IsEmpty (First))
        printf ("List is empty. No deletion performed!\n");
    else {
        p = First;
        First = p->next;

        Free_Node (p);
    }
}
```

XÓA NÚT GIỮA DANH SÁCH

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

```
void Delete_After (NODEPTR p)
{
    NODEPTR q;
    if (p == NULL || p->next == NULL)
        printf ("Cannot delete!\n");
    else {
        q = p->next;    // q is the node that will be deleted
        p->next = q->next;

        Free_Node (q);
    }
}
```

XÓA NÚT CUỐI DANH SÁCH

- Delete_End(): Xoá phần tử cuối DSLK
- Cài đặt

```
void DeleteEnd(NodePtr first)
{
    if(first==NULL)return;
    if(first->next==NULL)
    {
        free(first);
        first=NULL;
    }
    else
    {
        NodePtr q,r;
        r=first;
        while(r->next!=NULL)
        {
            q=r;
            r=r->next;
        }
        q->next=NULL;
        free(r);
    }
}
```

XÓA TOÀN BỘ DANH SÁCH

- Delete_All(): Xoá toàn bộ danh sách
- Cài đặt

```
void Delete_All (NODEPTR &First)
{
    NODEPTR p;
    while (First != NULL) // reach to end ?
    {
        p = First;
        First = First->next;    // *First = p->next;
        Free_Node (p);
    }
}
```

DUYỆT TOÀN BỘ DANH SÁCH

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

```
void Traverse (NODEPTR First)
```

```
{  
    NODEPTR p;  
    int count = 0;  
    p = First;  
    if (p == NULL)  
        printf ("List is empty!\n");  
    while (p != NULL)  
    { // reach to end ?  
        printf ("\n %5d%8d", count++, p->info);  
        p = p->next;  
    }  
}
```

TÌM KIẾM TRONG DANH SÁCH

- Search(): Tìm nút đầu tiên trong DS có info bằng với x. Nếu tìm thấy nút có (info == x) thì trả về địa chỉ của nút, nếu không, trả về NULL.

- Cài đặt:

```
NODEPTR Search (NODEPTR First, int x)
```

```
{    NODEPTR p;
```

```
    p = First;
```

```
    // not reach to end and not found
```

```
    while (p != NULL && p->info != x)
```

```
        p = p->next;
```

```
    return (p);
```

```
}
```


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

```
void Selection_Sort (NODEPTR *First)
{
    NODEPTR p, q, pmin;
    int min;
    for (p = *First; p->next != NULL; p = p->next) {
        min = p->info;
        pmin = p;
        for (q = p->next; q != NULL; q = q->next)
            if (min > q->info) {
                min = q->info;
                pmin = q;
            }
        // hoan doi truong info cua hai nut p va pmin
        pmin->info = p->info;
        p->info = min;
    }
}
```

NỘI DUNG

- Khái niệm
- Ưu điểm và hạn chế
- Các phép toán cơ bản
- Bài tập

NỘI DUNG

- Khái niệm
- Ưu điểm và hạn chế
- Các phép toán cơ bản
- Bài tập

BÀI TẬP

1. Viết chương trình con thêm một phần tử trong danh sách liên kết đã có thứ tự sao cho ta vẫn có một danh sách có thứ tự.
2. Viết chương trình con tìm kiếm và xóa một phần tử trong danh sách liên kết có thứ tự.
3. Viết chương trình con loại bỏ các phần tử trùng nhau (giữ lại duy nhất 1 phần tử) trong một danh sách liên kết có thứ tự không giảm
4. Viết chương trình con đảo ngược một danh sách liên kết

BÀI TẬP

5. Viết chương trình con xóa khỏi danh sách liên kết lưu trữ các số nguyên các phần tử là số nguyên lẻ
6. Viết chương trình con tách một danh sách liên kết chứa các số nguyên thành hai danh sách: một danh sách gồm các số chẵn còn cái kia chứa các số lẻ.
7. Để lưu trữ một số nguyên lớn, ta có thể dùng danh sách liên kết chứa các chữ số của nó. Hãy tìm cách lưu trữ các chữ số của một số nguyên lớn theo ý tưởng trên sao cho việc cộng hai số nguyên lớn là dễ dàng thực hiện. Viết chương trình con cộng hai số nguyên lớn

BÀI TẬP

8. Đa thức $P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ được lưu trữ trong máy tính dưới dạng một danh sách liên kết mà mỗi phần tử của danh sách là một bản ghi có ba trường lưu giữ hệ số, số mũ, và trường con trỏ để trỏ đến phần tử kế tiếp. Chú ý cách lưu trữ đảm bảo thứ tự giảm dần theo số mũ của từng hạng tử của đa thức.

- ❑ Hãy viết khai báo thực hiện được sự lưu trữ này.
- ❑ Dựa vào sự cài đặt ở trên, viết chương trình con thực hiện việc cộng hai đa thức.
- ❑ Viết chương trình con tính giá trị và lấy đạo hàm của đa thức.

BÀI TẬP

9. Đa thức $P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ được lưu trữ trong máy tính dưới dạng một mảng theo nguyên theo các cách sau:

Cách 1: Phần tử đầu tiên trong mảng lưu trữ bậc n của đa thức. $n + 1$ phần tử tiếp theo lần lượt lưu các hệ số từ a_n đến a_0

Cách 2: Phần tử đầu tiên trong mảng lưu trữ k là số các hệ số khác 0. $2k$ phần tử tiếp theo lưu trữ k cặp {hệ số, mũ} tương ứng

- ❑ Viết chương trình con thực hiện việc cộng hai đa thức.
- ❑ Viết chương trình con tính giá trị và lấy đạo hàm của đa thức.



Ngăn xếp



Ngăn xếp

- Mục đích, yêu cầu
 - Giới thiệu các yếu tố cơ bản của ngôn ngữ C
 - Sinh viên cần nắm được cách sử dụng biến, hằng, các cấu trúc điều khiển
- Hình thức tổ chức dạy học: lý thuyết: 2, thảo luận, bài tập: 3, thực hành, tự học: 10
- Thời gian: 6 tiết - tuần 11 và tuần 12
- Địa điểm: theo phân công P2

Nội dung

- Tổng quan về ngăn xếp
- Cài đặt ngăn xếp bằng mảng
- Cài đặt ngăn xếp bằng DSLK
- Bài tập

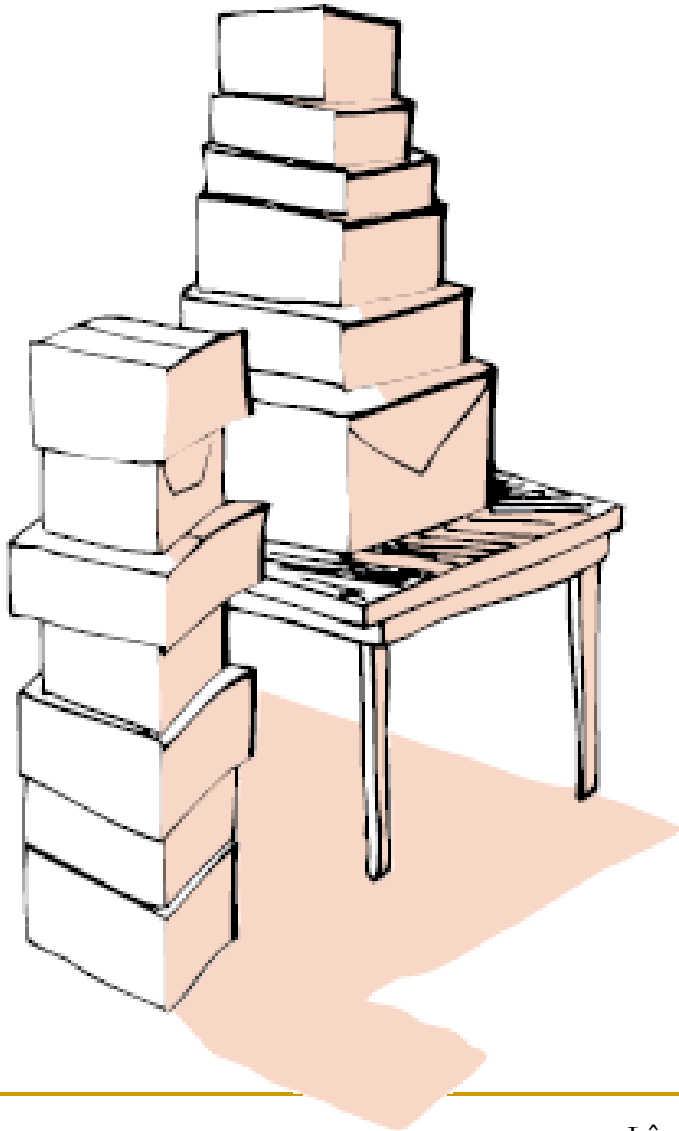
Các yếu tố cơ bản của ngôn ngữ C

- Mục đích, yêu cầu
 - Giới thiệu khái niệm stack, cài đặt và ứng dụng các phép toán stack bằng mảng, con trỏ sử dụng ngôn ngữ C
 - Sinh viên cần nắm được khái niệm stack, cài đặt và ứng dụng các phép toán stack bằng mảng, con trỏ sử dụng ngôn ngữ C
- Hình thức tổ chức dạy học: lý thuyết: 3, thảo luận, bài tập: 3, thực hành, tự học: 12
- Thời gian: tiết 5- tuần 1 và tuần 2
- Địa điểm: theo phân công P2

Nội dung

- Tổng quan về ngăn xếp
- Cài đặt ngăn xếp bằng mảng
- Cài đặt ngăn xếp bằng DSLK
- Bài tập

MÔ TẢ STACK



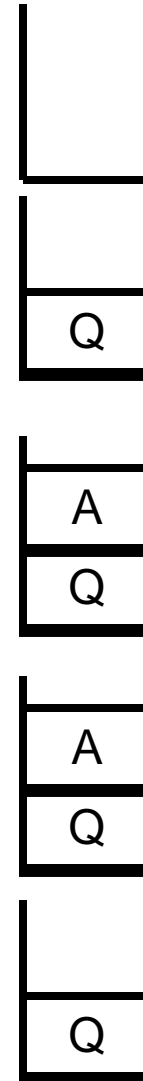
- Một stack là một cấu trúc dữ liệu mà việc thêm vào và loại bỏ được thực hiện tại một đầu (gọi là đỉnh – top của stack).
- Là một dạng vào sau ra trước – LIFO (Last In First Out)

CÁC PHÉP TOÁN CỦA STACK

- Khởi tạo Stack
- Kiểm tra Stack rỗng
- Kiểm tra Stack đầy
- Thêm một phần tử vào Stack
- Lấy một phần tử ra khỏi Stack

VÍ DỤ CÁC PHÉP TOÁN

- Stack rỗng:
- Đẩy (push) Q vào:
- Đẩy A vào:
- Lấy (pop) ra một => được A:
- Lấy ra một => được Q và stack rỗng:



NỘI DUNG

- Tổng quan về ngăn xếp
- Cài đặt ngăn xếp bằng mảng
- Cài đặt ngăn xếp bằng DSLK
- Bài tập

NỘI DUNG

- Tổng quan về ngăn xếp
- Cài đặt ngăn xếp bằng mảng
- Cài đặt ngăn xếp bằng DSLK
- Bài tập

KHAI BÁO STACK

```
const MAX=100;  
typedef struct  
{  
    int top;  
    float nut[MAX];  
} Stack;
```

KHỞI TẠO STACK

- Thao tác này thực hiện việc gán giá trị -1 cho biến top, cho biết ngăn xếp đang ở trạng thái rỗng.
- `void StackInitialize(Stack& s)`
{
 s.top=-1;
}

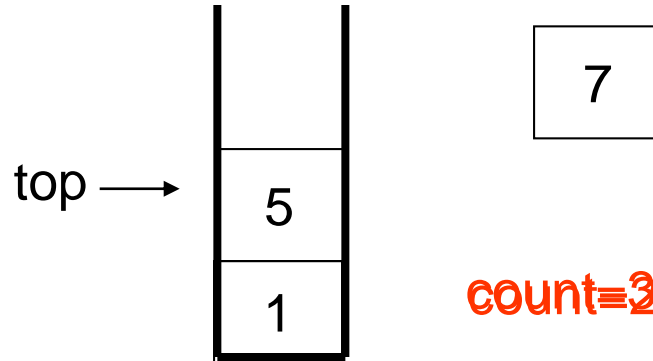
KIỂM TRA STACK RỖNG

- Stack rỗng khi $top = -1$
- `int StackEmpty(Stack s)`
{
 `return (s.top == -1);`
}

KIỂM TRA STACK ĐẦY

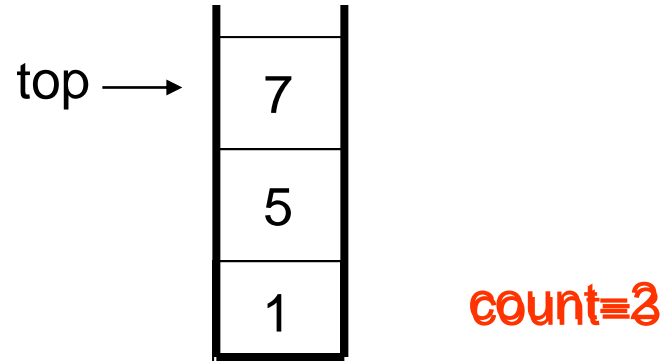
- Stack đầy khi $top = MAX - 1$
- `int StackFull(Stack s)`
{
 `return (s.top == MAX - 1);`
}

THÊM MỘT PHẦN TỬ VÀO STACK



```
void Push(Stack &s,float x)
{
    if(StackFull(s))
        printf("\nNgan xep da day!");
    else
        s.nut[++s.top]=x;
}
```

LẤY MỘT PHẦN TỬ TỪ STACK



```
float Pop(Stack &s)
{
    if(StackEmpty(s))
        printf("Ngan xep da rong!");
    else
        return s.nut[s.top--];
}
```


NỘI DUNG

- Tổng quan về ngăn xếp
- Cài đặt ngăn xếp bằng mảng
- Cài đặt ngăn xếp bằng DSLK
- Bài tập

NỘI DUNG

- Tổng quan về ngăn xếp
- Cài đặt ngăn xếp bằng mảng
- Cài đặt ngăn xếp bằng DSLK
- Bài tập

KHAI BÁO STACK

```
struct Node
{
    float info;
    struct Node *next;
};
typedef struct Node* StackNode;
typedef struct
{
    StackNode top;
}Stack;
```

KHỞI TẠO STACK

- Thao tác này thực hiện việc gán giá trị NULL cho biến top, cho biết ngăn xếp đang ở trạng thái rỗng.
- `void StackInitialize(Stack &s)`
{
 s.top=NULL;
}

KIỂM TRA STACK RỖNG

- Stack rỗng khi top=NULL
- int StackEmpty(Stack s)
{
 return s.top==NULL;
}

THÊM MỘT PHẦN TỬ VÀO STACK

```
void Push(Stack &s,float x)
{
    StackNode sn=(StackNode)malloc(sizeof(struct Node));
    sn->info=x;
    sn->next=s.top;
    s.top=sn;
}
```

LẤY MỘT PHẦN TỬ TỪ STACK

```
float Pop(Stack &s)
{
    if(StackEmpty(s))
        printf("\nStack is empty!");
    else
    {
        StackNode sn=s.top;
        float x=sn->info;
        s.top=s.top->next;
        free(sn);
        return x;
    }
}
```

NỘI DUNG

- Tổng quan về ngăn xếp
- Cài đặt ngăn xếp bằng mảng
- Cài đặt ngăn xếp bằng DSLK
- Bài tập

NỘI DUNG

- Tổng quan về ngăn xếp
- Cài đặt ngăn xếp bằng mảng
- Cài đặt ngăn xếp bằng DSLK
- Bài tập

BÀI TẬP

1. Viết chương trình đổi số nguyên không âm sang số nhị phân
2. Viết chương trình đảo ngược một xâu ký tự
3. Viết chương trình đảo ngược một danh sách
4. Cho một stack S. Hãy viết chương trình con thực hiện các công việc sau:
 - Đếm số phần tử của stack S
 - Xuất nội dung phần tử thứ n của stack S
 - Xuất nội dung của stack S
 - Loại phần tử thứ n của stack S

Hàng đợi

Hàng đợi

- Mục đích, yêu cầu
 - Giới thiệu khái niệm về hàng đợi, cài đặt hàng đợi bằng mảng, con trỏ và ứng dụng
 - Sinh viên cần nắm được khái niệm về hàng đợi, cài đặt hàng đợi bằng mảng, con trỏ và ứng dụng
- Hình thức tổ chức dạy học: lý thuyết: 3, thảo luận, bài tập:, thực hành, tự học: 3
- Thời gian: 3 tiết - tuần 13
- Địa điểm: theo phân công P2

Nội dung

- Tổng quan về hàng đợi
- Cài đặt hàng đợi bằng mảng
- Cài đặt hàng đợi bằng DSLK
- Bài tập

MÔ TẢ QUEUE

- Một queue là một cấu trúc dữ liệu mà việc thêm vào được thực hiện ở một đầu (rear) và việc lấy ra được thực hiện ở đầu còn lại (front)
- Phần tử vào trước sẽ ra trước – FIFO (First In First Out)



CÁC PHÉP TOÁN CỦA QUEUE

- Khởi tạo queue (*QueueInitialize*)
- Kiểm tra queue rỗng (*QueueEmpty*)
- Thêm một giá trị vào cuối của queue (*Put*)
- Bỏ giá trị đang có ở đầu của queue (*Get*)

NỘI DUNG

- Tổng quan về hàng đợi
- Cài đặt hàng đợi bằng mảng
- Cài đặt hàng đợi bằng DSLK
- Bài tập

NỘI DUNG

- Tổng quan về hàng đợi
- Cài đặt hàng đợi bằng mảng
- Cài đặt hàng đợi bằng DSLK
- Bài tập

KHAI BÁO

```
const max=30;  
struct Queue  
{  
    int head,tail,count;  
    float node[max];  
};
```

KHỞI TẠO

```
void QueueInitialize(Queue &q)
{
    q.count=0;
    q.head=q.tail=-1;
}
```

KIỂM TRA RỖNG

```
int QueueEmpty(Queue q)
{
    return q.count==0;
}
int QueueFull(Queue q)
{
    return q.count==max;
}
```

THÊM MỘT PHẦN TỬ

```
void Put(Queue &q,float x)
{
    if(QueueFull(q))
    {
        printf("\nQueue is full!");
        return;
    }
    else
    {
        if(QueueEmpty(q)
            q.head=0;
        if(q.tail==max-1)
            q.tail=0;
        else
            q.tail=(q.tail+1);
        q.node[q.tail]=x;
        q.count++;
    }
}
```

XÓA MỘT PHẦN TỬ

```
float Get(Queue &q)
{
    if(QueueEmpty(q))
        return 0;
    else
    {
        float x=q.node[q.head];
        q.head=(q.head+1)%max;
        q.count--;
        return x;
    }
}
```

NỘI DUNG

- Tổng quan về hàng đợi
- Cài đặt hàng đợi bằng mảng
- Cài đặt hàng đợi bằng DSLK
- Bài tập

NỘI DUNG

- Tổng quan về hàng đợi
- Cài đặt hàng đợi bằng mảng
- Cài đặt hàng đợi bằng DSLK
- Bài tập

KHAI BÁO

```
struct Node
{
    int info;
    struct Node *next;
};
typedef struct Node *QueueNode;
typedef struct
{
    QueueNode head ;
    QueueNode tail ;
}Queue;
```

KHỞI TẠO

```
void QueueInitialize(Queue &q)
{
    q.head=NULL;
    q.tail=NULL;
}
```

KIỂM TRA RỒI

```
int QueueEmpty(Queue q)
{
    return (q.head==NULL);
}
```

THÊM MỘT PHẦN TỬ

```
void Put(Queue &q,int x)
{
    QueueNode ql=(QueueNode)malloc(sizeof(struct Node));
    ql->info=x;
    ql->next=NULL;
    if(QueueEmpty(q))
    {
        q.head=q.tail=ql;
    }
    else
    {
        q.tail->next=ql;
        q.tail=ql;
    }
}
```

LẤY MỘT PHẦN TỬ

```
int Get(Queue &q)
{
    if(QueueEmpty(q))
        printf("\nQueue is Empty!");
    else
    {
        QueueNode temp=q.head;
        int x=temp->info;
        q.head=q.head->next;
        if(temp->next==NULL)
            q.tail=NULL;
        free(temp);
        return x;
    }
}
```

NỘI DUNG

- Tổng quan về hàng đợi
- Cài đặt hàng đợi bằng mảng
- Cài đặt hàng đợi bằng DSLK
- Bài tập

NỘI DUNG

- Tổng quan về hàng đợi
- Cài đặt hàng đợi bằng mảng
- Cài đặt hàng đợi bằng DSLK
- Bài tập

BÀI TẬP

1. Cài đặt hàng đợi bằng mảng và di chuyển khi mảng bị tràn
2. Cài đặt hàng đợi bằng mảng xoay vòng nhưng không xử dụng biến count
3. Dùng Queue kiểm tra một chuỗi ký tự có đối xứng không?
4. Cài đặt thuật toán duyệt đồ thị theo chiều rộng(BFS) sử dụng hàng đợi

File và các thao tác I/O

File và các thao tác I/O

- Mục đích, yêu cầu
 - Giới thiệu các thao tác với file, file nhị phân, file văn bản
 - Sinh viên cần nắm được các thao tác với file, file nhị phân, file văn bản
- Hình thức tổ chức dạy học: lý thuyết: 3, thảo luận, bài tập: 3, thực hành, tự học: 12
- Thời gian: 6 tiết - tuần 14 và tuần 15
- Địa điểm: theo phân công P2

Nội dung

- Đóng, mở file, kiểm tra lỗi
- Nhập xuất kí tự
- Các hàm nhập xuất theo kiểu văn bản
- Các hàm nhập xuất theo kiểu nhị phân
- Nhập xuất ngẫu nhiên và các hàm di chuyển con trỏ

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 gập 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 được

feof(); fscanf(); fprintf(); fflush()

- feof: cho biết đã đến cuối tập tin chưa (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

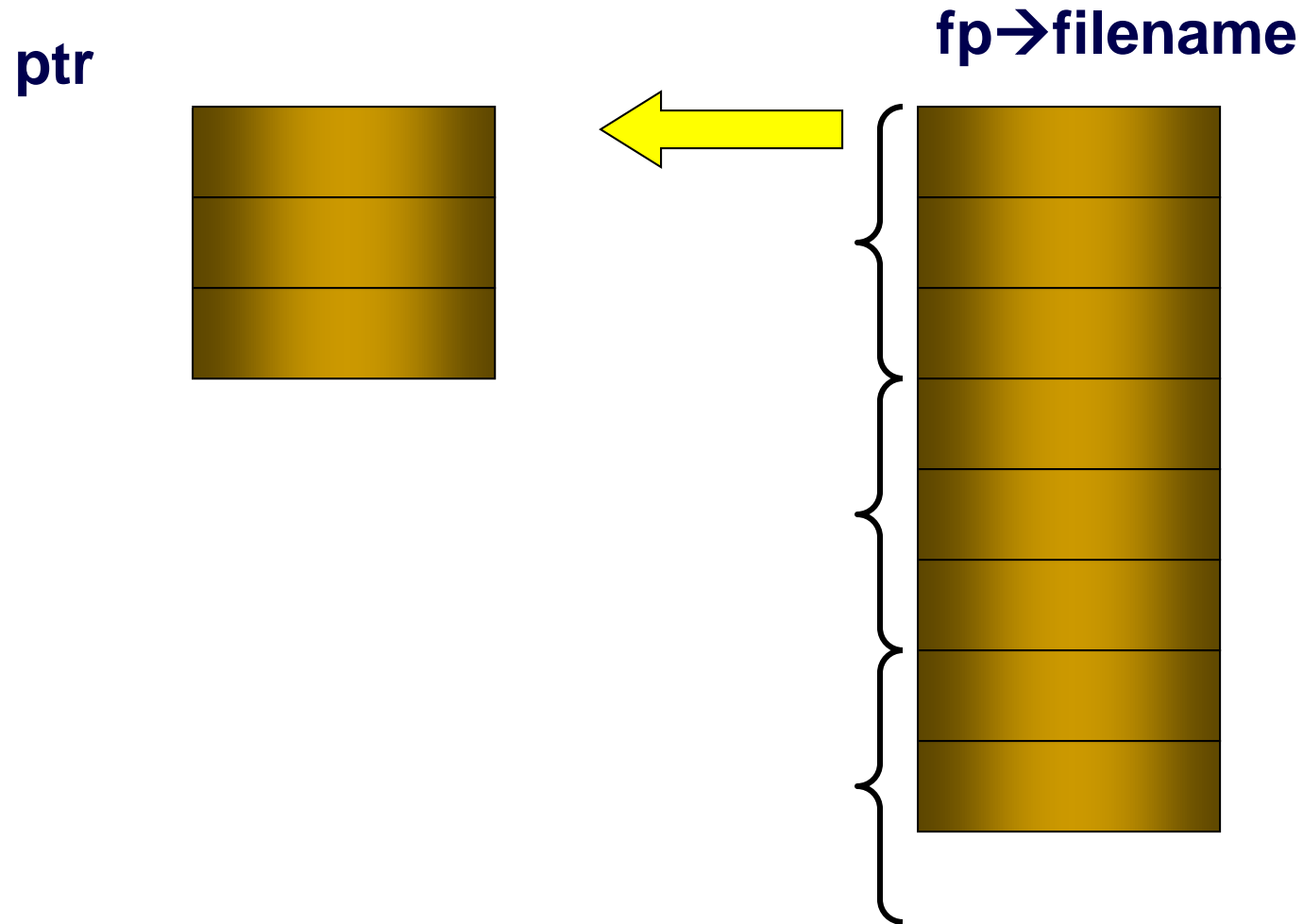
- fread() : đọc
size_t fread (void *Ptr, size_t ItemSize,
size_t NumItem, FILE * fp);

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 định 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 xóa

```
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(argv[i]);
7. printf (“ %d\n”, res);
8. }
```

■ G:\>addint 3 -2 1 5 6 -2 1  
12