CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

DATA STRUCTURE AND ALGORITHMS

Tài liệu học tập

- Giáo trình:
  - C & Data Structures, P. S. Deshpande, O. G. Kakde - CHARLES RIVER MEDIA, INC. Hingham, Massachusetts.
- Tham khảo:
  - Giáo trình Cấu trúc dữ liệu 1, Trần Hạnh Nhi – Dương Anh Đức, Trường ĐHKHTN – ĐHQG TP.HCM.
- Phần mềm lập trình:
  - C-Free
  - Borland C++

Nội dung môn học

- Chương 0: **Giới thiệu chung**
- Chương 1: **Ôn tập C/C++**
- Chương 2: **Đệ quy** (Recursion)
- Chương 3: **Tìm kiếm** (Searching)
- Chương 4: **Sắp xếp** (Sorting)
- Chương 5: **Ngăn xếp - Hàng đợi** (Stacks - Queues)
- Chương 6: **Danh sách liên kết** (Linked List)
- Chương 7: **Cây** (Tree)
- ÔN TẬP - KIỂM TRA (REVIEW – TEST)

Chương 0: **Giới thiệu chung**

Nội dung

- ✔Cấu trúc dữ liệu
- Thuật toán
- Độ phức tạp của thuật toán

Cấu trúc dữ liệu

- (1) Sự tổ chức hợp lý của các thành phần dữ liệu,
- (2) Tập các thao tác để truy cập các thành phần dữ liệu.

- (1) the logical arrangement of data elements, combined with
- (2) the set of operations we need to access the elements.

Ví dụ các cấu trúc dữ liệu

- Mảng (array)
- Danh sách liên kết (linked list)
- Ngăn xếp (stack)
- Hàng đợi (queue)
- Cây (tree)
- …

Nội dung

- Cấu trúc dữ liệu
- ✓Thuật toán
- Độ phức tạp của thuật toán

Thuật toán

- Tập các bước *có thể tính toán được* để đạt được kết quả mong muốn

- A *computable* set of steps to achieve a desired result

Ví dụ

- Tính tổng các số nguyên lẻ từ 1 □n
  - B1: S=0
  - B2: i=1
  - B3: Nếu i=n+1 thì sang B7, ngược lại sang B4
  - B4: S=S+i
  - B5: i=i+2
  - B6: Quay lại B3
  - B7: Tổng cần tìm là S

Mối quan hệ của CTDL và thuật toán

CTDL + Thuật toán = Chương trình

Ví dụ

- Một chương trình quản lý điểm thi của sinh viên cần lưu trữ các điểm số của 3 sinh viên. Giả sử mỗi sinh viên có 4 điểm số ứng với 4 môn học khác nhau, dữ liệu có dạng bảng như sau:

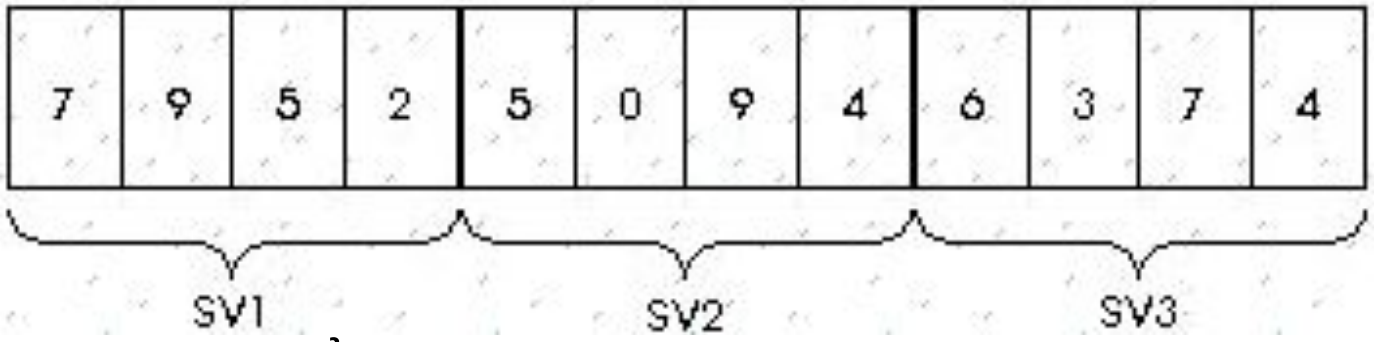
Sinh viên	Môn 1	Môn 2	Môn3	Môn4
SV 1	7	9	5	2
SV 2	5	0	9	4
SV 3	6	3	7	4

Ví dụ

- Chỉ xét thao tác xử lý là xuất điểm số các môn của từng sinh viên.
- Phương án 1 : Sử dụng mảng một chiều:

```
int result [12] = {7, 9, 5, 2, 5, 0, 9, 4, 6, 3, 7, 4};
```

  - các phần tử sẽ được lưu trữ như sau:



  - Truy xuất điểm số môn j của sinh viên i phải sử dụng một công thức xác định chỉ số tương ứng trong mảng result:

    result[(i*số cột) + j]

Ví dụ


```
void XuatDiem() //Xuất điểm số của tất cả sinh viên
{
   const int so_mon = 4;
   int  sv,mon;
   for (int i=0; i<12; i++)
   {
    sv = i/so_mon;
    mon = i % so_mon;
    cout<<"Điểm môn "<<mon<<" của sv "<<sv<<"là:"
    <<result[i];
   }
}
```

Ví dụ

- Chỉ xét thao tác xử lý là xuất điểm số các môn của từng sinh viên.
- Phương án 2 : Sử dụng mảng hai chiều:

```
int result[3][4] ={{ 7, 9, 5, 2}, { 5, 0, 9, 4}, { 6, 3, 7, 4 }};
```

  - các phần tử sẽ được lưu trữ như sau:

	Cột 0	Cột 1	Cột 2	Cột 3
Dòng 0	result[0][0] =7	result[0][1] =9	result[0][2] =5	result[0][3] =2
Dòng 1	result[1][0] =5	result[1][1] =0	result[1][2] =9	result[1][3] =4
Dòng 2	result[2][0] =6	result[2][1] =3	result[2][2] =7	result[2][3] =4

  - Truy xuất điểm số môn j của sinh viên i cũng chính là phần tử nằm ở vị trí (dòng i, cột j) trong mảng: result[i][j]

Ví dụ


```
void XuatDiem() //Xuất điểm số của tất cả sinh viên
{
    const int so_mon = 4, so_sv = 3;
    for ( int i=0; i<so_sv; i++)
        for ( int j=0; j<so_mon; j++)
            cout<<"Điểm môn "<< j <<" của sv "<< i <<"là:"
                result[i][j];
}
```

Nội dung

- Cấu trúc dữ liệu
- Thuật toán
- ✓ Độ phức tạp của thuật toán (algorithm complexity)

Độ phức tạp của thuật toán

- Phân tích thuật toán
  - Tính đúng
  - Tính đơn giản
  - Không gian
  - Thời gian chạy của thuật toán

Độ phức tạp của thuật toán

- Thời gian chạy của thuật toán
  - Đánh giá như thế nào
    - Thực nghiệm
    - Xấp xỉ

Độ phức tạp của thuật toán

- Thực nghiệm
  - Chịu sự hạn chế của ngôn ngữ lập trình
  - Ảnh hưởng bởi trình độ của người cài đặt
  - Chọn được các bộ dữ liệu thử đặc trưng cho tất cả tập các dữ liệu vào của thuật toán: khó khăn và tốn nhiều chi phí
  - Phụ thuộc nhiều vào phần cứng

Độ phức tạp của thuật toán

- Xấp xỉ tiệm cận
  - Cách thông dụng nhất để đánh giá một thuật toán là ký hiệu tiệm cận gọi là Big-O
  - *Định nghĩa toán học của Big-O:*

    Cho f và g là hai hàm từ tập các số nguyên hoặc số thực đến số thực. Ta nói f(x) là O(g(x)) nếu tồn tại hằng số C và k sao cho: $|f(x)| \leq C\,|g(x)|$ *với mọi* $x > k$
  - Ví dụ, hàm $f(x) = x^2 + 3x + 2$ là $O(x^2)$

    Thật vậy, khi $x > 2$ thì $x < x^2$ và $2 < 2x^2$

    Do đó $x^2 + 3x + 2 < 6x^2$

Độ phức tạp của thuật toán

- Một số kết quả Big-O quan trọng:
  - Hàm đa thức:
    - $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$
    - Khi đó f(x) là $O(x^n)$
  - Hàm giai thừa:
    - $f(n) = n!$ là $O(n^n)$
  - Logarit của hàm giai thừa:
    - $f(n) = \log n!$ là $O(n \log n)$
  - Hàm điều hòa
    - $H(n) = 1 + 1/2 + 1/3 + .. + 1/n$ là $O(\log n)$

Độ phức tạp của thuật toán

- Một số lớp thuật toán

n \ Hàm	n	lg n	Nlgn	n^2	n^3	2^n
1	1	0	0	1	1	2
2	2	1	2	4	8	4
4	n	2	8	16	64	16
8	8	3	24	64	512	256
16	16	4	64	256	4096	65536
32	32	5	160	1024	32768	2,147,483,648

Độ phức tạp của thuật toán

- Một số lớp thuật toán

$O(\log_2 n)$
$O(n)$
$O(n\log_2 n)$
$O(n^2)$
$O(n^k)$

} Độ phức tạp đa thức $\Rightarrow$ chấp nhận được

$O(2^n)$
$n!$

} Độ phức tạp cao $\Rightarrow$ khả chấp nhận

Độ phức tạp của thuật toán

- Một số lớp thuật toán

Độ phức tạp	Thuật ngữ/tên phân lớp
O(1)	Độ phức tạp hằng số
O(logn)	Độ phức tạp logarit
O(n)	Độ phức tạp tuyến tính
O(nlogn)	Độ phức tạp nlogn
$O(n^a)$	Độ phức tạp đa thức
$O(a^n)$, a > 1	Độ phức tạp hàm mũ
O(n!)	Độ phức tạp giai thừa

Độ phức tạp của thuật toán

- Ví dụ, xét hàm sau:
  - Hai lệnh *cout* ngoài vòng lặp có độ phức tạp hằng O(1) – vì không phụ thuộc vào N
  - Số lệnh *cout* trong vòng lặp bằng với kích thước mảng, do đó vòng lặp có độ phức tạp O(N)
  - Tổng cộng: Hàm f có độ phức tạp: 2 * O(1) + O(N)
- Độ phức tạp: O(N)

```
void f (int a[], int n)
{
      cout<< "N = "<< n;
      for (int i = 0; i < n; i++ )
            cout<<a[i];
      cout<< "\n";
}
```

Chương 1: Ôn tập C/C++

(Tham khảo tài liệu môn Phương Pháp Lập Trình)

Chương 1: Ôn tập C/C++

1. ✓ Cấu trúc chương trình C/C++
2. Các cú pháp cơ bản
3. Địa chỉ (Address)
4. Con trỏ (Pointer)
5. Mảng (Array)
6. Mảng con trỏ (Pointer array)
7. Mảng hai chiều (Two-dimensional array)
8. Cấu trúc (Structure)
9. Con trỏ cấu trúc (Structure pointer)
10. Chuỗi (String)
11. Tập tin (File)
12. Hàm (Function)

1. Cấu trúc chương trình C/C++

Cấu trúc chương trình C

```
#include “stdio.h”
#include “conio.h”

void main()  /*ham chinh*/
{
    int a=7;
    printf( “%d”, a );
    getch();
}
```

1. Cấu trúc chương trình C/C++

Cấu trúc chương trình C++

```
#include “iostream.h”
#include “conio.h”

void main()  /*ham chinh*/
{
    int a=7;
    cout<< a ;
    getch();
 }
```

1. Cấu trúc chương trình C/C++

- Qui cách viết chương trình
  - Các dòng trong cùng một khối thẳng cột
  - Khối con của một khối lùi vào ít nhất một TAB
  - Sử dụng thống nhất các qui tắc giãn cách
  - Ghi chú thích ở những chỗ cần thiết

Chương 1: Ôn tập C/C++

1. Cấu trúc chương trình C/C++
2. ✔ Các cú pháp cơ bản
3. Địa chỉ (Address)
4. Con trỏ (Pointer)
5. Mảng (Array)
6. Mảng con trỏ (Pointer array)
7. Mảng hai chiều (Two-dimensional array)
8. Cấu trúc (Structure)
9. Con trỏ cấu trúc (Structure pointer)
10. Chuỗi (String)
11. Tập tin (File)
12. Hàm (Function)

2. Các cú pháp cơ bản

- Khai báo biến:

```
Kiểu_dữ_liệu  tên_biến;
```

- Khai báo và khởi tạo biến:

```
Kiểu_dữ_liệu  tên_biến = giá trị;
```

- Khai báo hằng số:

```
const  Kiểu_dữ_liệu  tên_biến = giá trị;
```

2. Các cú pháp cơ bản

Các kiểu dữ liệu cơ sở

Kiểu	Phạm vi biểu diễn	Kích thước (byte)
char	-2^7 □ 2^7-1	1
int	-2^{15} □ $2^{15}-1$	2
long	-2^{31} □ $2^{31}-1$	4
float	3.4E-38 □ 3.4E+38	4
double	1.7E-308 □ 1.7E+308	6

2. Các cú pháp cơ bản

Các phép toán số học

Phép toán	Tên
+	cộng
-	trừ
*	nhân
/	chia
%	chia lấy phần dư
++, --	Phép tăng, giảm 1

2. Các cú pháp cơ bản

Các phép toán so sánh

Phép toán	Tên
>	lớn hơn
>=	lớn hơn hoặc bằng
<	nhỏ hơn
<=	nhỏ hơn hoặc bằng
==	bằng
!=	khác

2. Các cú pháp cơ bản

Các phép toán logic

Phép toán	Tên gọi
&&	AND
||	OR
!	NOT

2. Các cú pháp cơ bản

- **Chuyển đổi kiểu:**
  - Trong biểu thức: kiểu thấp hơn sẽ được nâng thành kiểu cao hơn trước khi thực hiện phép toán
  - Ví dụ:
    - 7 + 3.5

2. Các cú pháp cơ bản

- **Chuyển đổi kiểu:**
  - Trong phép gán: Giá trị của biểu thức vế phải được chuyển sang kiểu của biến vế trái

Ví dụ:

```
int  a=5/2; //a=?
float  b=5/2;    //b=?
```

2. Các cú pháp cơ bản

- **Ép kiểu:**
  - Cú pháp:

    (Kiểu) biểu_thức
  - Ví dụ:

    (float) 23

    (int) x

    float(23)

    x*1.0

2. Các cú pháp cơ bản

Các toán tử điều khiển:

- if
- if … else …
- switch … case …
- for
- while
- do … while

2. Các cú pháp cơ bản

Toán tử điều kiện

```
if ( bt_điều_kiện )
   khối_lệnh;
```

```
if ( bt_điều_kiện )
    khối_lệnh_1
else
    khối_lệnh_2
```

```
if ( bt_điều_kiện_1 )
    khối_lệnh_1
else if ( bt_điều_kiện_2 )
    khối_lệnh_2
…
else
    khối_lệnh_n
```

2. Các cú pháp cơ bản

Câu lệnh switch

```
switch (biểu_thức_nguyên) {
  case hằng_1:
         các_câu_lệnh_1
         [break;]
  case hằng_2:
         các_câu_lệnh_2
         [break;]
  ...
  [default:
         các_câu_lệnh]
}
```

2. Các cú pháp cơ bản

Các toán tử điều khiển lặp

```
for (bt_khởi_tạo;
  bt_kiểm_tra; bt_tăng)
  khối_lệnh
```

```
while ( bt_điều_khiển )
{
 câu_lệnh_1;
 câu_lệnh_2;
 …
}
```

```
do
{
 khối_lệnh
}while (bt_điều_khiển);
```

2. Các cú pháp cơ bản

- Cấp phát bộ nhớ tĩnh:
  - Ví dụ: int a,b;
- Cấp phát bộ nhớ động:
  - Bộ nhớ cũng có thể được cấp phát tại thời gian chạy, gọi là **Cấp phát động** (dynamic allocation)
  - Dùng toán tử **new** để cấp phát bộ nhớ động
  - Ví dụ:

```
int *P;
P = new int;
```

2. Các cú pháp cơ bản


```
//Program to demonstrate pointers
//and dynamic variables
#include <iostream>
int main()
{
    int *p1, *p2;
    p1 = new int;
    *p1 = 10;
    p2 = p1;
    cout << "*p1 = " << *p1 << endl;
    cout << "*p2 = " << *p2 << endl << endl;

    *p2 = 30;
    cout << "*p1 = " << *p1 << endl;
    cout << "*p2 = " << *p2 << endl << endl;

    p1 = new int;
    *p1 = 40;
    cout << "*p1 = " << *p1 << endl;
    cout << "*p2 = " << *p2 << endl << endl;
}
```

```
*p1 = 10
*p2 = 10
```

```
*p1 = 30
*p2 = 30
```

```
*p1 = 40
*p2 = 30
```

2. Các cú pháp cơ bản

- Cấp phát bộ nhớ động:
  - **heap:** vùng bộ nhớ đặc biệt dành riêng cho các biến động. Để tạo một biến động mới, hệ thống cấp phát không gian từ heap. Nếu không còn bộ nhớ, **new** không thể cấp phát bộ nhớ thì nó trả về giá trị **NULL**
  - Trong lập trình, ta nên luôn kiểm tra lỗi này:

```
int *p;
p = new int;
if (p == NULL) {
    cout << "Loi cap phat bo nho\n";
    exit;
```

2. Các cú pháp cơ bản

- Hủy bộ nhớ động:
  - Trả lại vùng bộ nhớ trỏ bởi **P**, nhưng không sửa giá trị của **P**
  - Dùng toán tử delete để hủy bộ nhớ động
  - Sau khi thực thi **delete**, giá trị của con trỏ không xác định
  - Ví dụ:

```
delete P;
```

Bài tập

- Viết chương trình tính diện tích, chu vi hình chữ nhật
- Viết chương trình tính diện tích, chu vi hình tròn

Chương 1: Ôn tập C/C++

1. Cấu trúc chương trình C/C++
2. Các cú pháp cơ bản
3. ✓ Địa chỉ (Address)
4. Con trỏ (Pointer)
5. Mảng (Array)
6. Mảng con trỏ (Pointer array)
7. Mảng hai chiều (Two-dimensional array)
8. Cấu trúc (Structure)
9. Con trỏ cấu trúc (Structure pointer)
10. Chuỗi (String)
11. Tập tin (File)
12. Hàm (Function)

3. Địa chỉ (Address)

- Mỗi biến đều có 2 thuộc tính: địa chỉ (address) và giá trị (value)

Trong bộ nhớ:

+ Tại địa chỉ 3: giá trị là 45

+ Tại địa chỉ 2: giá trị là "Dave"

1	4096
2	"Dave"
3	45
4	"Matt"
5	95.5
6	"wbru"
7	0
8	"zero"

- Lấy địa chỉ của biến: dùng **&**

```
int y=90;
cout << "Value of 'y' is: " << y << "\n";
cout << "Address of 'y' is: " << &y << "\n\n";
```

Chương 1: Ôn tập C/C++

1. Cấu trúc chương trình C/C++
2. Các cú pháp cơ bản
3. Địa chỉ (Address)
4. ✓ Con trỏ (Pointer)
5. Mảng (Array)
6. Mảng con trỏ (Pointer array)
7. Mảng hai chiều (Two-dimensional array)
8. Cấu trúc (Structure)
9. Con trỏ cấu trúc (Structure pointer)
10. Chuỗi (String)
11. Tập tin (File)
12. Hàm (Function)

4. Con trỏ (Pointer)

- Là một biến mà giá trị của nó chứa một địa chỉ
- Định nghĩa một con trỏ: thêm dấu * vào trước tên biến
- Ví dụ: `int *ia;`

  `int x, *p, *q;`

- Toán tử * : trả về nội dung của địa chỉ được chứa trong một biến con trỏ

4. Con trỏ (Pointer)

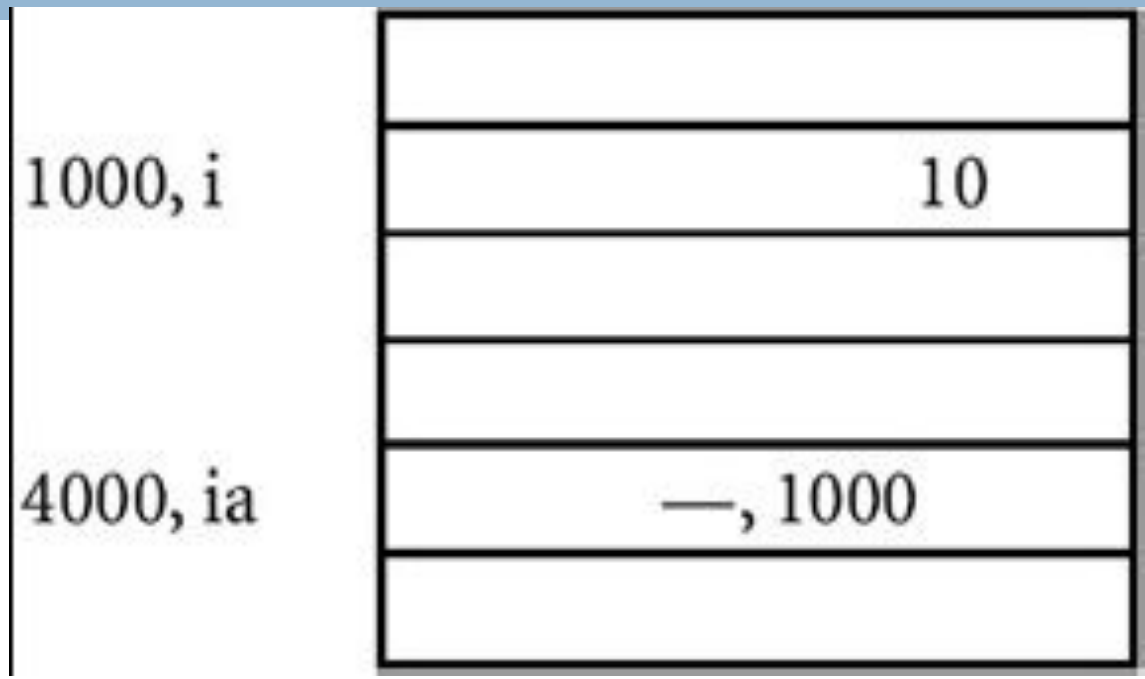
- Các phép toán số học trên con trỏ:
  - Phép gán
  - Phép cộng, trừ một con trỏ với một số nguyên
  - Tăng, giảm
  - Ví dụ:

```
int x, *p, *q;
p = &x;
p = p+2;  // tăng 2 kích thước bộ nhớ int
q = p;
```

4. Con trỏ (Pointer)


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

void main ()
{
    int i;
    int *ia;
    i = 10;
    ia = &i;
    cout<<" The address of i is "<< ia <<"\n";
    cout<<" The value at that location is "<< i <<"\n";
    cout<<" The value at that location is "<< *ia <<"\n";
    *ia = 50;
    cout<<" The value of i is "<<i;
}
```



4. Con trỏ (Pointer)


```
int i;
int *ia;
cout<<"Dia chi cua i "<< &i << " co gia tri ="<<i <<endl;
cout<<" Dia chi cua ia " << &ia << " co gia tri = " << ia<<endl;
i = 10;
ia = &i;
cout<<"sau khi gan gia tri:"<<endl;
cout<<" Dia chi cua i "<< &i << " co gia tri ="<<i <<endl;
cout<<" Dia chi cua ia " << &ia << " co gia tri= " << ia<<
                " tro đen: "<< *ia;
```

```
#include <iostream>
int main()
{   int x = 10; int y = 20;
    int *p1, *p2;

    p1 = &x;
    p2 = &y;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    cout << "*p1 = " << *p1 << endl;
    cout << "*p2 = " << *p2 << endl << endl;

    *p1 = 50;
    *p2 = 90;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    cout << "*p1 = " << *p1 << endl;
    cout << "*p2 = " << *p2 << endl << endl;

    p1 = p2;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    cout << "*p1 = " << *p1 << endl;
    cout << "*p2 = " << *p2 << endl << endl;
}
```

```
x = 10
y = 20
*p1 = 10
*p2 = 20
```

```
x = 50
y = 90
*p1 = 50
*p2 = 90
```

```
x = 50
y = 90
*p1 = 90
*p2 = 90
```

Chương 1: Ôn tập C/C++

1. Cấu trúc chương trình C/C++
2. Các cú pháp cơ bản
3. Địa chỉ (Address)
4. Con trỏ (Pointer)
5. ✔ Mảng (Array)
6. Mảng con trỏ (Pointer array)
7. Mảng hai chiều (Two-dimensional array)
8. Cấu trúc (Structure)
9. Con trỏ cấu trúc (Structure pointer)
10. Chuỗi (String)
11. Tập tin (File)
12. Hàm (Function)

5. Mảng (Array)

- Mảng:
  - Là một cấu trúc dữ liệu
  - Là danh sách các phần tử có cùng kiểu
  - Cho phép truy xuất ngẫu nhiên đến các phần tử của nó thông qua chỉ số mảng

	[0]	[1]	[2]	[3]	[4]
A	12	2	8	5	1

- **Kiểu_dữ_liệu tên_mảng[số_phần_tử];**

5. Mảng (Array)

- Địa chỉ của mỗi phần tử trong mảng:
  - Mỗi phần tử trong mảng có một địa chỉ trong bộ nhớ

    (Each element of the array has a memory address)
- Ví dụ:

```
void printdetail (int a[])
{
 for(int i = 0; i<5; i++)
 {
        cout<< "value in array "<< a[i] <<" at address: " << &a[i];
 }
```

5. Mảng (Array)

- Truy cập mảng sử dụng con trỏ:
  - Có thể truy cập từng phần tử của mảng bằng cách sử dụng con trỏ

```
void print_usingptr(int a[], int n)
{
    int *b;
    b=a;
    cout<<"value in array\n";
    for (int i=0; i<n; i++)
    {
      cout<<*b<<"  ";
      b++;
    }
}
```

```
void print_usingptr(int *a, int n)
{
      cout<<"value in array\n";
      for (int i=0; i<n; i++)
      {
       cout<<*(a+i)<<"   ";
      }
}
```

5. Mảng (Array)

- **Cấp phát động cho mảng và hủy mảng:**
  - Kích thước của mảng động không cần là hằng số mà có thể có giá trị được quyết định tại thời gian chạy
  - **new T[n]** : cấp phát một mảng gồm **n** đối tượng kiểu **T** và trả về một con trỏ tới đầu mảng
  - **delete [] p** : hủy mảng mà **p** trỏ tới

    P *phải* trỏ tới đầu mảng động, nếu không, kết quả của **delete** sẽ phụ thuộc vào trình biên dịch và loại dữ liệu đang sử dụng. Ta có thể nhận được lỗi *runtime error* hoặc kết quả sai

5. Mảng (Array)


```
#include <iostream>
int main ()
{
   int size;
   cin << size;
   int* A = new int[size];      // dynamically allocate array

   A[0] = 0; A[1] = 1; A[2] = 2;
   cout << "A[1] = " << A[1] << endl;

   delete [] A;                 // delete the array
}
```

5. Mảng (Array)


```
#include <iostream>

int main ()
{
   int* A = new int[6];      // dynamically allocate array

   A[0] = 0; A[1] = 1; A[2] = 2;
   A[3] = 3; A[4] = 4; A[5] = 5;

   int *p = A + 2;
   cout << "A[1] = " << A[1] << endl;

   delete [] p;              // illegal!!!

   // results depend on particular compiler
   cout << "A[1] = " << A[1] << endl;
}
```

Chương 1: Ôn tập C/C++

1. Cấu trúc chương trình C/C++
2. Các cú pháp cơ bản
3. Địa chỉ (Address)
4. Con trỏ (Pointer)
5. Mảng (Array)
6. ✓ Mảng con trỏ (Pointer array)
7. Mảng hai chiều (Two-dimensional array)
8. Cấu trúc (Structure)
9. Con trỏ cấu trúc (Structure pointer)
10. Chuỗi (String)
11. Tập tin (File)
12. Hàm (Function)

6. Mảng con trỏ (Pointer array)

- Có thể khai báo mảng con trỏ (tương tự như mảng số)
- Trong mảng con trỏ, các phần tử mảng lưu con trỏ

```
void main()
{
        int *a[5];
        int i1=4, i2=3, i3=2, i4=1, i5=0;
        a[0] = &i1;
        a[1] = &i2;
        a[2] = &i3;
        a[3] = &i4;
        a[4] = &i5;
        printarr(a);
}
```

```
void printarr(int *a[])
{
        for(int j=0; j<5; j++)
        {
                cout<<a[j] <<"   ";
        }
}
```

Chương 1: Ôn tập C/C++

1. Cấu trúc chương trình C/C++
2. Các cú pháp cơ bản
3. Địa chỉ (Address)
4. Con trỏ (Pointer)
5. Mảng (Array)
6. Mảng con trỏ (Pointer array)
7. ✓ Mảng hai chiều (Two-dimensional array)
8. Cấu trúc (Structure)
9. Con trỏ cấu trúc (Structure pointer)
10. Chuỗi (String)
11. Tập tin (File)
12. Hàm (Function)

7. Mảng hai chiều (Two-dimensional array)

- Khai báo:

Kiểu_dữ_liệu tên_mảng[số_dòng] [số_cột];

- Ví dụ:

```
int a[2][3];
float  mang[10][5];
```

- Không lấy địa chỉ của phần tử mảng hai chiều bằng toán tử **&**

7. Mảng hai chiều (Two-dimensional array)

- Duyệt mảng hai chiều sử dụng con trỏ:
  - Ví dụ:

```
float *pa, a[2][3];
pa = (float*)a; // neu khong ep kieu thi C se canh bao
                // nhung chuong trinh van chay tot
```

Khi đó pa trỏ tới a[0][0]

pa+1 trỏ tới a[0][1]

pa+2 trỏ tới a[0][2]

pa+3 trỏ tới a[1][0]

pa+4 trỏ tới a[1][1]

pa+5 trỏ tới a[1][2]

Bài tập

- Viết chương trình cho nhập 1 mảng hình chữ nhật và tính diện tích, chu vi của chúng
- Viết chương trình cho nhập 1 mảng hình tròn và tính diện tích, chu vi của chúng

Chương 1: Ôn tập C/C++

1. Cấu trúc chương trình C/C++
2. Các cú pháp cơ bản
3. Địa chỉ (Address)
4. Con trỏ (Pointer)
5. Mảng (Array)
6. Mảng con trỏ (Pointer array)
7. Mảng hai chiều (Two-dimensional array)
8. ✔ Cấu trúc (Structure)
9. Con trỏ cấu trúc (Structure pointer)
10. Chuỗi (String)
11. Tập tin (File)
12. Hàm (Function)

8. Cấu trúc (Structure)

- Được sử dụng khi cần thao tác trên dữ liệu có nhiều thuộc tính
- Cú pháp:

```
struct  Tên_kiểu_cấu_trúc {
    các_thành_phần;
};
```

- Ví dụ:

```
struct Ngay {
        int thang;
        int ngay;
        int nam;
};
```

8. Cấu trúc (Structure)

- Khai báo biến kiểu cấu trúc (2 cách):

Tên_cấu_trúc tên_biến;

struct Tên_cấu_trúc tên_biến;

- Ví dụ:

Ngay n;

hoặc: struct Ngay ng;

- Khởi tạo cho một cấu trúc:

Ví dụ:

```
struct Ngay d={10,15,2004};
struct Ngay a[]={   {10,15,2004},
{10,16,2004},                    {10,17,2004},
            {10,18,2004}};
```

8. Cấu trúc (Structure)

- Truy cập thành phần của cấu trúc:
  - Dùng toán tử "**.**": Tên_biến_cấu_trúc**.**Tên_thành _phần
  - Ví dụ:

```
Ngay d;
d.ngay = 10;
d.thang = 10;
d.nam = 1910;
```

8. Cấu trúc (Structure)

- Ví dụ

```
struct student
{
    char name[30];
    float marks;
};

main ( )
{
    student student1;
    char s1[30];float f;
    cin>>student1.name;
    cin>>student1.marks;

    cout<<"Name is:"<<student1.name;
    cout<< "Marks are:" << student1.marks;
}
```

Bài tập

- Viết chương trình tính diện tích, chu vi hình chữ nhật (yêu cầu khai báo cấu trúc hình chữ nhật)
- Viết chương trình tính diện tích, chu vi hình tròn (yêu cầu khai báo cấu trúc hình tròn)

Chương 1: Ôn tập C/C++

1. Cấu trúc chương trình C/C++
2. Các cú pháp cơ bản
3. Địa chỉ (Address)
4. Con trỏ (Pointer)
5. Mảng (Array)
6. Mảng con trỏ (Pointer array)
7. Mảng hai chiều (Two-dimensional array)
8. Cấu trúc (Structure)
9. ✔ Con trỏ cấu trúc (Structure pointer)
10. Chuỗi (String)
11. Tập tin (File)
12. Hàm (Function)

9. Con trỏ cấu trúc (Structure pointer)

- Giống như các kiểu dữ liệu khác, ta có thể khai báo con trỏ cấu trúc
- Ví dụ

```
struct Ngay *p, a[10], *p1, b;
p = a; p1 = &b;
```

- Truy nhập thành phần của cấu trúc:

```
tên_con_trỏ->tên_thành_phần
```

hoặc :

```
(*tên_con_trỏ).tên_thành_phần
```

9. Con trỏ cấu trúc (Structure pointer)

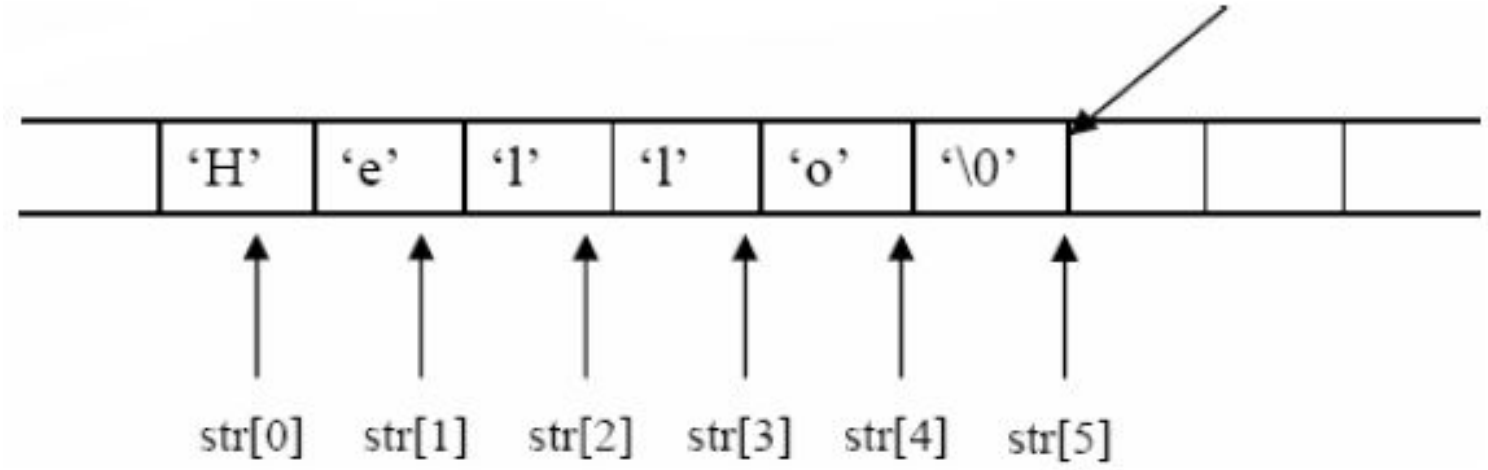

```
#include<iostream.h>
struct student
{
        char name[30];
        float marks;
};
void main ( )
{
        student *sv;
        char s[30];
        float f;
        cin>>s;
        cin>>f;
        sv->name = s;
        sv->marks = f;
        cout<<" Name is "<< sv->name<<"\n";
        cout<<" Marks are "<<sv->marks<<"\n";
}
```

Chương 1: Ôn tập C/C++

1. Cấu trúc chương trình C/C++
2. Các cú pháp cơ bản
3. Địa chỉ (Address)
4. Con trỏ (Pointer)
5. Mảng (Array)
6. Mảng con trỏ (Pointer array)
7. Mảng hai chiều (Two-dimensional array)
8. Cấu trúc (Structure)
9. Con trỏ cấu trúc (Structure pointer)
10. ✔ Chuỗi (String)
11. Tập tin (File)
12. Hàm (Function)

10. Chuỗi (String)

- Là mảng các ký tự (array of char)
- Kết thúc bởi ký tự null "**\0**" (ending with null char \0)
- Chuỗi hằng được tự động thêm "\0"
- Ví dụ: char str[]="Hello";



10. Chuỗi (String)

- Khai báo chuỗi:
  - char str[] = {‘H’,’e’,’l’,’l’,’o’,’\0’};
  - char str[] = “Hello”;
  - char *str = “Hello”;

10. Chuỗi (String)

- Hàm nhập chuỗi:
  - **char *gets(char *s);**
  - Nhận ký tự cho đến khi nhận dấu Enter
  - Tự động thêm ký tự '\0'
  - So sánh với cin>>s; //????
- Hàm xuất chuỗi:
  - **int puts(const char *s);**
  - cout<<s;

10. Chuỗi (String)

- Keyboard buffer

```
char szKey[] = "aaa";
char s[10];
do {
        cout<<"doan lai di?";
        gets(s);
} while (strcmp (szKey,s) != 0);
puts ("OK. corect");
```

If user input: aaaaaaaaaaaaa???

10. Chuỗi (String)

- Một số hàm về chuỗi: (cần #include <string.h>)
  - strcpy(s1, s2)
  - strcat(s1, s2)
  - strlen(s1)
  - strcmp(s1, s2) -> (-1,0,1)
  - strchr(s1, ch)
  - strstr(s1, s2)
  - …

10. Chuỗi (String)

- Ví dụ:

```
char s1[80], s2[80];
cout << "Input the first string: :";
gets(s1);
cout << "Input the second string: ";
gets(s2);
cout << "Length of s1= " << strlen(s1);
cout << "Length of s2= " << strlen(s2);
if (!strcmp(s1, s2))
    cout << "These strings are equal\n";
strcat(s1, s2);
cout << "s1 + s2: " << s1 << endl;;
strcpy(s1, "This is a test.\n");
cout << s1;
if (strchr(s1, 'e'))  cout << "e is in " << s1;
if (strstr(s2, "hi"))  cout << "found hi in " <<s2;
```

Chương 1: Ôn tập C/C++

1. Cấu trúc chương trình C/C++
2. Các cú pháp cơ bản
3. Địa chỉ (Address)
4. Con trỏ (Pointer)
5. Mảng (Array)
6. Mảng con trỏ (Pointer array)
7. Mảng hai chiều (Two-dimensional array)
8. Cấu trúc (Structure)
9. Con trỏ cấu trúc (Structure pointer)
10. Chuỗi (String)
11. ✔ Tập tin (File)
12. Hàm (Function)

11. Tập tin (File)

- Cần #include <io.h>
- Tạo file mới:
  - FILE *fp;
  - fp=fopen(“d:\\test.txt", "wb"))
- Ghi nội dung vào file:
  - fwrite(&Address, sizeof(TYPE), count, fp);
- Đóng file (Lưu file):
  - fclose(fp);

11. Tập tin (File)

- FILE *f;
- File text, File binary
- fopen, fclose, fread, fwrite, fscanf, fprintf, feof….

11. Tập tin (File)

- Đọc file:

```
FILE *fp;
fp = fopen(“d:\\test.txt", “rb"))
while (fwrite(&Address, sizeof(TYPE), count, fp))
{
        // xử lý nội dung đọc được
}
fclose(fp);
```

Chương 1: Ôn tập C/C++

1. Cấu trúc chương trình C/C++
2. Các cú pháp cơ bản
3. Địa chỉ (Address)
4. Con trỏ (Pointer)
5. Mảng (Array)
6. Mảng con trỏ (Pointer array)
7. Mảng hai chiều (Two-dimensional array)
8. Cấu trúc (Structure)
9. Con trỏ cấu trúc (Structure pointer)
10. Chuỗi (String)
11. Tập tin (File)
12. ✓ Hàm (Function)

12. Hàm (Function)

- Các cú pháp định nghĩa hàm:

```
void Tên_Hàm (kiểu_dữ_liệu tên_biến,…)
{
      // các khai báo biến,…
      // các lệnh…
}
```

```
Kiểu_dữ_liệu Tên_Hàm (kiểu_dữ_liệu
tên_biến,…)
{
      // các khai báo biến,…
      // các lệnh…
      return value;
}
```

12. Hàm (Function)

- Cách gọi hàm:

1. Chuẩn bị các tham số để gởi cho hàm nếu có:
   - Khai báo biến tương ứng và cho nhập dữ liệu cho biến (nếu cần)
2. Hàm không trả về giá trị (void):
   - Tên_Hàm (tham_số_1, tham_số_2,…);

2. Hàm có trả về giá trị:
   - Khai báo một *biến* có kiểu trùng với kiểu trả về của hàm
   - Viết lệnh gán: *biến* = Tên_Hàm (tham_số_1, tham_số_2,…);
   - Sử dụng *biến* để xuất, tính toán, gọi hàm khác

12. Hàm (Function)

- Nguyên mẫu hàm (`Prototype`)
  - Nguyên mẫu hàm được sử dụng để khai báo một hàm nhờ đó nó có thể được sử dụng trong chương trình trước khi hàm đó được định nghĩa thực sự
  - Công dụng:
    - Giúp chương trình mạch lạc
    - Giúp tổ chức chương trình
    - Cho phép sử dụng hàm trước khi định nghĩa

```
#include <stdio.h>
int compute_sum (int n);  /* Function
  Prototype*/
void main() {
   int lim = 8, sum;
   cout<<"Main lim (before call) is
     "<<lim<<"\n";
   sum = compute_sum(lim);
   cout<<"Main lim (after call) is "<<lim<<"\n";
   cout<<"The sum of integers from 1 to "<<
     lim<< " is "<<sum;
}
int compute_sum(int n) {
   int sum=0;
   for(;n>0;n--)
     sum +=n;
   return sum;
}
```

```
#include <stdio.h>

int compute_sum(int n) {
    int sum=0;
    for(;n>0;n--)
      sum +=n;
    return sum;
}
void main() {
    int lim = 8, sum;
    cout<<"Main lim (before call) is
      "<<lim<<"\n";
    sum = compute_sum(lim);
    cout<<"Main lim (after call) is "<<lim<<"\n";
    cout<<"The sum of integers from 1 to "<<
      lim<< " is "<<sum;
}
```