LẬP TRÌNH NÂNG CAO

Bài 10+11+12: Kiểu cấu trúc (struct) và kiểu hợp nhất (union)

Nội dung chính

1. Các kiểu dữ liệu tự tạo
2. Cấu trúc (struct)
    1. Khai báo
    2. Phép toán
    3. Trường bit
    4. Kích cỡ của struct
3. Bài tập struct
4. Hợp nhất (union)
5. Liệt kê (enum)
6. Cấu trúc tự trỏ và danh sách
7. Bài tập

Phần 1

Các kiểu dữ liệu tự tạo

Các kiểu dữ liệu tự tạo

- Kiểu dữ liệu: quy định về loại dữ liệu được ghi trong biến
  - Có tính quy ước, vì dữ liệu luôn được ghi ở dạng byte / bit
  - Xác định cách thức xử lý giá trị khi tham gia tính toán
- Các ngôn ngữ lập trình cung cấp một số kiểu dữ liệu cơ bản (số nguyên, số thực, logic,…)
  - Cũng là các kiểu dữ liệu thường dùng trong cuộc sống
- Cuộc sống có nhiều kiểu dữ liệu phức tạp hơn:
  - Phân số: tử số (số thực) + mẫu số (số thực)
  - Số phức: phần thực (số thực) + phần ảo (số thực x *i*)
  - Ngày: ngày (số nguyên) + tháng (số nguyên) + năm (số nguyên)
  - Giờ: giờ (số nguyên) + phút (số nguyên) + giây (số nguyên) +…
  - Thời gian: Ngày + Giờ

Tự tạo kiểu dữ liệu mới

- Các kiểu dữ liệu phức tạp thường là tổ hợp từ các loại cơ bản và đôi khi từ những loại tổ hợp đơn giản hơn
- Không ngôn ngữ lập trình nào cung cấp được mọi loại dữ liệu cần thiết cho mọi nhu cầu thực tiễn
- C/C++ cung cấp các cơ chế cho phép lập trình viên tự tạo các kiểu dữ liệu mới
  - Cấu trúc (struct)
  - Hợp nhất (union)
  - Liệt kê (enum)
  - Lớp (class)
- Chúng ta thực tế đã sử dụng rất nhiều các kiểu dữ liệu tự tạo (string, vector,…)

Phần 2

Cấu trúc (struct)

Đặt vấn đề: xét một bài toán cụ thể

- Quản lý dữ liệu thời gian (ngày tháng năm)
- Cách làm thông thường: bộ 3 biến lưu ngày, tháng, năm
  - Mỗi khi có dữ liệu ngày tháng thì cần khai báo thêm cả 3
- Truyền thông tin vào hàm? Dùng 3 tham số
- Trả về kết quả từ hàm? Không làm được
- Nhận xét
  - Đặt tên biến nhập nhằng, khó quản lý, phải có quy tắc riêng
  - Truyền tham số cho hàm dài dòng
  - Tìm kiếm, sắp xếp, sao chép,… khó khăn
- Giải pháp của C/C++: Gom những thông tin này tạo thành một kiểu dữ liệu mới

Khai báo struct

- Cú pháp sử dụng struct:

```
struct <tên kiểu> {
    <các dữ liệu thành phần>
};
```

- Sau khi khao báo struct, ta có một kiểu dữ liệu mới, trùng tên với struct
- Ví dụ:

```
struct ThoiGian {           // kiểu dữ liệu ThoiGian
    int ngay, thang, nam;   // các thành phần con
};
ThoiGian homnay;            // biến kiểu ThoiGian
homnay.ngay  = 3;           // thành phần ngày = 3
homnay.thang = 4;           // thành phần tháng = 4
homnay.nam   = 2021;        // thành phần năm = 2021
```

Khai báo struct

```
// khai báo kiểu ĐIỂM và 2 biến
struct DIEM {
    int x;
    int y;
} diem1, diem2;

// khai báo biến riêng rẽ
struct DIEM diem3, diem4;

// khai báo biến trong C++ được bỏ struct
DIEM diem5, diem6;
```

Khai báo struct

```
// khai báo kiểu gián tiếp bằng typedef
typedef struct {
    int x;
    int y;
} POINT;

// khai báo biến với định kiểu typedef
struct POINT p1, p2;

// khai báo và khởi trị
struct POINT {
    int x;
    int y;
} p3 = { 123, 456 }, p4;
```

Phép toán với struct

- Một kiểu struct được xem là một kiểu dữ liệu mới
- Phép gán hai biến struct cho nhau: sao chép nguyên văn dữ liệu từ nguồn tới đích
- Phép truy cập thành phần:
  - Mỗi thành phần của struct là một biến độc lập
  - Truy cập vào từng thành phần thông qua toán tử thành phần (.)

```
struct Top10 {
    double top[10];
    int len;
};
Top10 abc;              // biến kiểu Top10
abc.top[0] = 9.9;       // gán thành phần
abc.len = 1;            // gán thành phần
Top10 xyz = abc;        // gán 2 struct
```

Phép toán với struct

- Truy cập struct lồng nhau

```
struct A { int x, y; };
struct B { A p, q; };
B b;
b.p.x = 1, b.p.y = 2;
```

- Toán tử thành phần sử dụng với con trỏ:

```
struct Top10 {
    double top[10];
    int len;
    int *end;
};
Top10 * t1 = new Top10;
t1->top[0] = 9.5;
(*t1).len = 1;
t1->end = nullptr;
```

Trường bit

- C/C++ cho phép chỉ định kích cỡ của một thành phần tới số bit

```
struct bit_fields {
    unsigned int b0 : 1;
    unsigned int b1 : 4;
    unsigned int b2 : 1;
    unsigned int b3 : 10;
};
```

- Trong ví dụ trên thì b0 chỉ dùng 1 bit, b1 dùng 4 bit, b2 dùng 1 bit và b3 dùng 10 bit
- Tất cả 4 trường trên có thể được “nén” vào trong một vùng 16 bit (2 byte)

Trường bit

- Trường bit là cách sử dụng tinh tế các dữ liệu tới đơn vị nhỏ nhất của máy tính, cách làm này rất thông dụng trong việc lập trình điều kiển hoặc giao tiếp cấp thấp
- Quy tắc:
  1. Số bit không được vượt quá số bit thực sự của kiểu khai báo
  2. Không thể thực hiện phép lấy địa chỉ (&) của trường bit
  3. Có thể ép trình dịch chuyển trường tiếp theo ra đầu byte bằng cách thêm một trường không tên cỡ 0 bit
  4. Biến nguyên luôn có một bit dấu, vì vậy rất thận trọng với trường bit dạng nguyên nhưng cỡ 1 bit
  5. Trường bit không thể khai báo dạng static
  6. Trường bit không thể khai báo dạng mảng

Trường bit

```
struct date {
    int d : 5;  // int hay unsigned int?
    int m : 4;  // int hay unsigned int?
    int y;
};

struct align {
    unsigned int x : 5;
    unsigned int : 0;       // cái gì đây?
    unsigned int y : 8;
};
```

Kích cỡ của struct

```
struct A {
    int a;
    double b;
};

struct B {
    int a;
    int b;
    double c;
};

struct C {
    int a;
    double c;
    int b;
};
```

Kích cỡ của struct

- Cơ chế canh biên:
  - Các biến thuộc struct sẽ được đẩy dịch theo từng bước, không nhất thiết phải đặt liên tiếp nhau trong bộ nhớ
  - Lợi: tăng tốc độ xử lý
  - Hại: kích cỡ của struct có thể tăng do xuất hiện nhiều ô nhớ thừa không được sử dụng
  - Nguy hiểm: không có sự nhất quán về mã máy
- Chỉ thị: #pragma pack(n)
  - N có thể là 1, 2, 4, 8, 16
  - Visual C++ mặc định là 8
  - Borland C++ mặc định là 1

Phần 3

Bài tập struct

Hãy tự tạo vài kiểu dữ liệu mới

1. Kiểu dữ liệu Point mô tả một điểm trên mặt phẳng tọa độ (gồm tọa độ trục X và trục Y)
2. Kiểu dữ liệu Line mô tả một đoạn thẳng trên mặt phẳng tọa độ (gồm 2 điểm đầu cuối)
3. Kiểu dữ liệu GiaoVien, lưu trữ thông tin về các giáo viên trong trường, gồm có: họ tên, địa chỉ cư trú, số điện thoại, năm vào trường
4. Kiểu dữ liệu SinhVien, lưu trữ thông tin về các sinh viên trong trường, gồm: họ tên, khóa học, lớp quản lý, địa chỉ, điện thoại

Bài tập

5. Xây dựng kiểu dữ liệu mới "Phân Số"

- Khai báo kiểu dữ liệu phân số (PhanSo) với tử số và mẫu số là các số nguyên
- Viết hàm nhập phân số
- Viết hàm chuyển phân số sang string (để in ra màn hình)
- Viết các hàm tính tổng, hiệu, tích, thương hai phân số
- Viết hàm kiểm tra phân số tối giản hay không
- Viết hàm tối giản phân số
- Viết hàm chuyển từ kiểu double sang phân số và ngược lại
- Viết hàm so sánh hai phân số
- Viết hàm so sánh phân số và số double

Bài tập

6.Xây dựng kiểu dữ liệu mới "Ngày"

- Khai báo kiểu dữ liệu ngày (Ngay)
- Viết hàm nhập dữ liệu ngày (ngày, tháng, năm)
- Viết hàm chuyển ngày sang string dạng dd-mm-yyyy
- Viết hàm kiểm tra ngày có thuộc năm nhuận không
- Viết hàm tính số thứ tự ngày trong năm
- Viết hàm tính số thứ tự ngày kể từ khi bắt đầu công lịch (ngày khởi điểm là ngày 1/1/1)
- Viết hàm cộng thêm k ngày (k có thể âm)
- Viết hàm tính khoảng cách giữa hai ngày
- Viết hàm so sánh hai ngày

Phần 4

Hợp nhất (union)

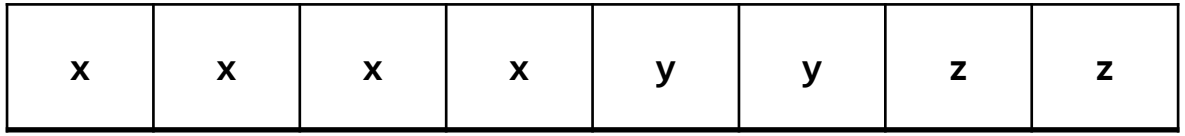
Hợp nhất (union)

- Hợp nhất là một kiểu đóng gói dữ liệu thú vị trong C/C++
- Nhắc lại về cấu trúc (struct):
  - Nhóm các mảnh dữ liệu liên quan với nhau thành một khối
  - Các dữ liệu con là các biến độc lập nhau, nằm trong khối nhớ cấp cho struct theo thứ tự khai báo
  - Thay đổi dữ liệu con này không ảnh hưởng đến dữ liệu con khác
- Hợp nhất (union):
  - Khai báo và mọi thứ (phép toán, cú pháp) giống hệt struct
  - Khác biệt duy nhất: các dữ liệu con nằm chồng lên nhau trong cùng một khối nhớ
  - Thay đổi dữ liệu con này có thể ảnh hưởng đến dữ liệu con khác

Hợp nhất (union)

```
#include <iostream>
using namespace std;
struct A {
    int x;
    short y;
    short z;
};
union B {
    int x;
    short y;
    short z;
};
int main() {
    cout << sizeof(A) << endl;      // 8
    cout << sizeof(B) << endl;      // 4
}
```

Sơ đồ bộ nhớ của struct A



Sơ đồ bộ nhớ của union B

Hợp nhất (union)

- Cú pháp khai báo của union: giống struct

```
union <tên kiểu> {
    <các dữ liệu thành phần>
};
```

- Mọi thành phần của union đều nằm chồng lên nhau (chung địa chỉ là phần đầu của union)
- Kích cỡ của union = kích cỡ của thành phần lớn nhất
- Vấn đề chia sẻ khối nhớ: cập nhật thành phần này ảnh hưởng đến thành phần khác
- Vậy union dùng vào việc gì?
  - Cung cấp nhiều kiểu nhìn cho một khối dữ liệu
  - Sử dụng cùng một khối dữ liệu theo các cách uyển chuyển hơn

Hợp nhất (union)

```
#include <iostream>
using namespace std;

union ABC {
    int *diachi;    // một con trỏ
    int giatri;     // nhìn con trỏ như là một số nguyên
};
int main() {
    int k = 100;
    ABC n;
    n.diachi = &k;
    cout << n.diachi << endl;   // góc nhìn là con trỏ
    cout << n.giatri << endl;   // góc nhìn là số nguyên
    cout << &k << endl;         // địa chỉ của k để so sánh
    cout << (int) &k << endl;   // đia chỉ của k ở số nguyên
}
```

Struct trong union

```
union Date {
    char str[11];
    struct {
        char day[2];
        char separator1;
        char month[2];
        char separator2;
        char year[4];
    } parts;
} ngaynghi = {"30/04/2021"};
```

Sơ đồ bộ nhớ của union Date

3	0	/	0	4	/	2	0	2	1	\0

Union trong struct

```
#include <iostream>
using namespace std;

struct Domain {
    string name;
    union {
        unsigned char p[4];
        unsigned int v;
    } ip;
} mydomain = {"txnam.net", { 150, 95, 105, 231 } };

int main() {
    cout << "Ten mien: " << mydomain.name << endl;;
    cout << "IP: " << mydomain.ip.v;
}
```

Phần 5

Liệt kê (enum)

Liệt kê (enum)

- Kiểu liệt kê là tập hợp các hằng số nguyên được đặt tên
- Cú pháp:

```
enum <tên kiểu> { <danh sách các hằng số> };
```

- Ví dụ:

```
// kiểu Ngày, liệt kê các ngày trong tuần
// giá trị các hằng số do trình dịch tự chọn
enum Ngay { chunhat, thu2, thu3, thu4, thu5, thu6, thu7 };

// kiểu Giới tính, liệt kê các giới tính được khai báo
// xác định luôn vài giá trị của hằng
// hằng số KhongKhaiBao được tự động chọn là 3
enum GioiTinh { Nam = 1, Nu = 2, KhongKhaiBao };

// kiểu Bài tập, liệt kê các loại bài tập được giao
enum BaiTap { mot_tiet, giua_ky, cuoi_ky, do_an };
```

Liệt kê (enum)

- Thay vì dùng kiểu liệt kê, lập trình viên có thể tự định nghĩa các hằng số
- Nhưng kiểu liệt kê gom các hằng số có cùng mối quan hệ thành một nhóm, giúp viết mã trong sáng, dễ hiểu hơn,
- Ví dụ:

```
GioiTinh abc = Nam;
switch (abc) {
    case Nam: cout << "Gioi tinh Nam"; break;
    case Nu:  cout << "Gioi tinh Nu"; break;
    case KhongKhaiBao:
        cout << "Khong khai gioi tinh"; break;
    default:
        cout << "Loi";
}
```

Phần 6

Cấu trúc tự trỏ và danh sách

Cấu trúc tự trỏ: khái niệm

- Cấu trúc tự trỏ là dạng struct mà trong bản thân struct chứa (các) con trỏ đến chính cấu trúc đó
- Cấu trúc bình thường vs Cấu trúc tự trỏ

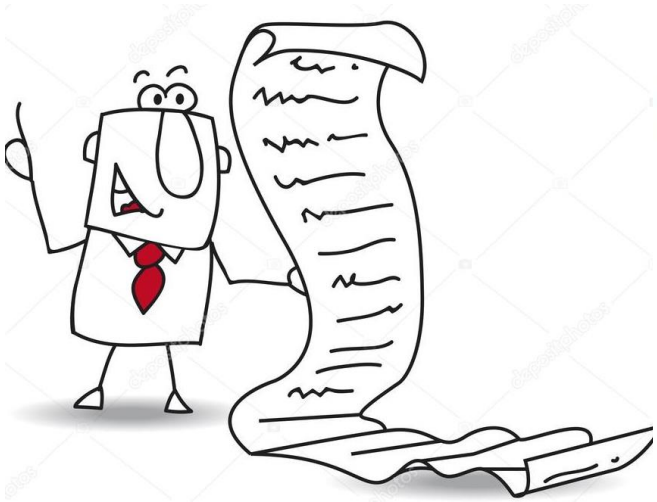
```
// một cấu trúc thông thường
struct Info {
    string name;    // tên
    int * data;     // con trỏ đến dữ liệu
    string *sub;    // một con trỏ khác
};
// nếu struct Node chứa một (vài) con trỏ đến Node
// thì đó là cấu trúc tự trỏ
struct Node {
    string student;     // trường tên sinh viên
    Node * next;        // con trỏ đến một Node
};
```

Cấu trúc tự trỏ: có nhiều loại

```
// cấu trúc tự trỏ dùng cho danh sách liên kết
struct ListNode {
    int data;           // dữ liệu
    ListNode *next;     // nút tiếp theo
};
// cấu trúc tự trỏ dùng cho cây nhị phân
struct BinaryTreeNode {
    int data;                           // dữ liệu
    BinaryTreeNode *left, *right;       // nút trái và nút phải
};
// cấu trúc tự trỏ dùng cho cây tổng quát
struct TreeNode {
    int data;               // dữ liệu
    int childNo;            // số nút con
    TreeNode **child;       // các nút con
};
```

Cấu trúc tự trỏ: ứng dụng vào danh sách (list)

- Danh sách: là cấu trúc dữ liệu thông dụng trong máy tính
  - Lấy cảm hứng từ cuộc sống
  - Các dữ liệu móc nối với nhau thành một chuỗi
  - Có thể dễ dàng xóa một mục
  - Có thể dễ dàng chèn thêm một mục
  - Duyệt theo thứ từ từ mục đầu tiên đến hết



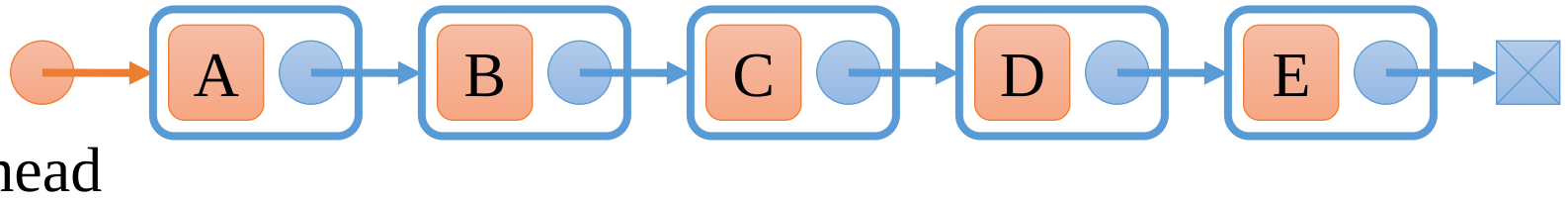
Cấu trúc tự trỏ: ứng dụng vào danh sách (list)

- Có nhiều cách cài đặt danh sách, sử dụng cấu trúc tự trỏ trong bài này chỉ là một trong những cách cài đặt, thường gọi là danh sách liên kết đơn (single linked list)
- Một nút trên danh sách: dùng cấu trúc tự trỏ

```
struct Node {
    string data;      // trường dữ liệu
    Node * next;      // con trỏ đến nút tiếp theo
};
```

- Khai báo danh sách: một con trỏ trỏ tới đầu danh sách

```
Node * head;      // con trỏ đến nút đầu tiên
```



Cấu trúc tự trỏ: ứng dụng vào danh sách (list)

```
// kiểm tra xem danh sách có rỗng không?
bool isEmpty(Node * head) {
    return nullptr == head;
}

// tạo một nút mới với dữ liệu value, con trỏ tiếp là next
Node * createNode(string value, Node * next = nullptr) {
    return new Node { value, next };
}

// tạo một danh sách chứa n giá trị lấy từ mảng value
Node * createList(string value[], int n) {
    Node * p = nullptr;
    for (int i = n-1; i >= 0; i--)
        p = createNode(value[i], p);
    return p;
}
```

Cấu trúc tự trỏ: ứng dụng vào danh sách (list)

```
// tạo một danh sách rỗng
Node * createList() {
    return nullptr;
}

// hủy một danh sách
void deleteList(Node * & head) {
    Node *ptr = head;
    while (nullptr != ptr) {
        Node * c = ptr;
        ptr = ptr->next;
        delete c;
    }
    head = nullptr;
}
```

Cấu trúc tự trỏ: ứng dụng vào danh sách (list)

```
// đếm số nút trên danh sách
int lengthList(Node * head) {
    int len = 0;
    while (nullptr != head) {
        len++;
        head = head->next;
    }
    return len;
}
// in ra các phần tử trong danh sách
void printList(Node * head) {
    while (nullptr != head) {
        cout << head->data << endl;
        head = head->next;
    }
}
```

Cấu trúc tự trỏ: ứng dụng vào danh sách (list)

```
// chèn giá trị value thành vị trí thứ n trong danh sách
// chèn vào cuối nếu n > số phần tử trong danh sách
void insertNode(Node * & head, string value, int n) {
    Node **ptr = &head;
    while (n > 0) {
        if (nullptr == *ptr) break;
        ptr = &((*ptr)->next);
        n--;
    }
    *ptr = createNode(value, *ptr);
}
```

Cấu trúc tự trỏ: ứng dụng vào danh sách (list)

```
// xóa giá trị thứ n trong danh sách
// không làm gì nếu n > số phần tử trong danh sách
void deleteNode(Node * & head, int n) {
    if (nullptr == head) return;
    Node **ptr = &head;
    for (int i = 0; i < n; i++) {
        ptr = &((*ptr)->next);
        if (nullptr == *ptr) return;
    }
    Node * c = *ptr;
    *ptr = c->next;
    delete c;
}
```

Phần 7

Bài tập

Bài tập phần struct và union

1. Sử dụng union Date được định nghĩa ở trang 27, hãy viết hàm nhập dữ liệu vào Date bằng cách nhập ngày tháng năm ở dạng số nguyên rồi chuyển thành dãy kí tự và ghi vào các trường day, month, year.
2. Sử dụng union Date được định nghĩa ở trang 27. Hãy viết hàm addYear làm nhiệm vụ cộng thêm 1 năm nữa vào dữ liệu hiện thời. In ra kết quả thực hiện.
3. Sử dụng struct Domain định nghĩa ở trang 28. Hãy viết hàm nhập dữ liệu gồm tên domain và 4 trường địa chỉ dạng IP4, sau đó in thông tin về domain ra màn hình dạng <tên domain> (<địa chỉ ip>).

Bài tập phần danh sách liên kết: viết các hàm

1. Hàm insertHead(value) chèn value vào đầu danh sách
2. Hàm insertTail(value) chèn value thành phần tử cuối cùng của danh sách
3. Hàm insertBeforeTail(value) chèn value thành phần tử đứng trước phần tử cuối cùng của danh sách
4. Hàm removeHead() xóa phần tử đầu của danh sách
5. Hàm removeTail() xóa phần tử cuối trong danh sách
6. Hàm removeAll(k) xóa mọi phần tử có giá trị k trong danh sách, hàm trả về số nút bị xóa khỏi danh sách
7. Hàm findNode(k) tìm và trả về con trỏ đến phần tử đầu tiên có giá trị k trong danh sách, trả về nullptr nếu không tìm được