**CÂU HỎI LÝ THUYẾT**

**Câu 1: Khi nào chương trình con được viết theo thủ tục, theo hàm và cách viết của từng loại?**

Chương trình con (CTC) là một đoạn chương trình thực hiện trọn vẹn hay một chức năng nào đó. Trong Turbo Pascal, có 2 dạng CTC:

* Thủ tục (PROCEDURE): Dùng để thực hiện một hay nhiều nhiệm vụ nào đó.
* Hàm (FUNCTION): Trả về một giá trị nào đó (có kiểu vô hướng, kiểu string). Hàm có thể sử dụng trong các biểu thức.

Ngoài ra, trong Pascal còn cho phép các CTC lồng vào nhau.

***Chú ý*: Trong quá trình xây dựng CTC, khi nào thì nên dùng thủ tục/hàm?**

|  |  |
| --- | --- |
| Dùng hàm | **Dùng thủ tục** |
| - Kết quả của bài toán trả về ***1 giá trị duy nhất (kiểu vô hướng, kiểu string).***  - Lời gọi CTC cần nằm trong các biểu thức tính toán. | - Kết quả của bài toán ***không trả về giá trị nào*** hoặc trả về ***nhiều giá trị***hoặc***trả về  kiểu dữ liệu có cấu trúc (Array, Record, File).***  - Lời gọi CTC không nằm trong các biểu thức tính toán. |

**Câu 2: Tham số có mấy loại? Khi nào dùng tham trị, khi nào dùng tham biến? Cái nào tốn bộ nhớ hơn? Cho ví dụ?**

**Tham trị** trong C++ không làm thay đổi giá trị của tham số thực tế được truyền vào hàm. Tức là khi truyền giá trị của tham số thực tế vào hàm, thì giá trị của tham số thực tế sẽ được copy cho tham số hình thức của hàm, giá trị của tham số hình thức và tham số thực tế ở hai bộ nhớ khác nhau, vì vậy thay đổi giá trị bên trong hàm không làm thay đổi giá trị của tham số thực tế truyền vào.

**Tham chiếu** trong C++ sẽ làm thay đổi giá trị của biến truyền vào hàm bởi vì chúng ta truyền vào địa chỉ của nó. Ở đây, địa chỉ của tham số thực tế được truyền vào hàm, vì vậy địa chỉ của tham số hình thức và tham số thực tế sẽ cùng địa chỉ trong bộ nhớ, cho nên khi thay đổi giá trị của tham số hình thức bên trong hàm, sẽ làm thay đổi giá trị gốc của tham số thực tế được truyền vào hàm.

**Chúng ta cùng tổng hợp điểm khác nhau giữa tham trị và tham chiếu trong C++ như trong bảng sau:**

|  |  |
| --- | --- |
| **Tham tri** | **Tham chiếu** |
| Một bản sao giá trị của biến được truyền vào hàm | Địa chỉ của biến được truyền vào hàm |
| Thay đổi biến bên trong hàm không làm thay đổi biến truyền vào hàm | Thay đổi biến bên trong hàm sẽ làm thay đổi biến truyền vào hàm |
| Tham số hình thức và tham số thực tế khác địa chỉ trong bộ nhớ | Tham số hình thức và tham số thực tế cùng địa chỉ trong bộ nhớ |

**Khi nào dùng tham trị tham biến:**

* Dùng tham trị khi bạn không muốn giá trị của biến bị thay đổi.
* Dùng tham chiếu khi bạn muốn hàm thay đổi giá trị biến của bạn.
* Dùng tham chiếu sẽ tiết kiệm bộ nhớ hơn, vì nó không phải tạo ra 1 bản sao.
* Nếu bạn dùng tham chiếu, bạn cũng không muốn hàm thay đổi giá trị của bạn thì bạn cần thêm từ khóa const vào tham số đó. Ví dụ: int f(const int &x).
* Nếu tham số là mảng => truyền theo tham chiếu. Chúng ta sẽ tìm hiểu nó trong chương sau.

**Câu 3: khác nhau giữa mảng và DSLK**

|  |  |
| --- | --- |
| **Mảng** | **Danh sách liên kết** |
| Vùng nhớ của các phần tử trong mảng được sắp xếp liên tục nhau | Vùng nhớ của các phần tử trong danh sách liên kết được sắp xếp tùy ý (do hệ điều hành). Các phần tử lưu 1 con trỏ trỏ tới phần tử tiếp theo. |
| Truy cập tới phần tử trong mảng là truy cập trực tiếp dựa vào chỉ số (ví dụ: a[0], a[1], a[2],…, a[n]) | Cần phải duyệt tuần tự khi muốn truy cập tới phần tử trong danh sách liên kết. |
| * Kích thước của mảng là hằng số, không thay đổi khi chạy chương trình * Sử dụng mảng không tối ưu được bộ nhớ. Có thể thừa hoặc thiếu bộ nhớ khi xóa hoặc chèn phần tử vào mảng | * Kích thước của danh sách liên kết có thể thay đổi khi chạy chương trình. * Sử dụng danh sách liên kết tối ưu được bộ nhớ. Vùng nhớ được cấp phát thêm khi cần chèn thêm phần tử mới, vùng nhớ được free khi xóa phần tử. |

### Bảng so sánh

| **So sánh** | **Array** | **Linked list** |
| --- | --- | --- |
| Cơ bản | Là một tập cố định các phần tử | Một tập hợp lớn các dữ liệu được sắp xếp theo thứ tự nào đó |
| Kích thước | Cố định, được định nghĩa lúc khai báo | Không cần định nghĩa, nó tự lớn lên và thu gọn lại lúc thực thi |
| Nơi lưu trữ | Địa điểm của các thành phần được lưu trữ trong suốt thời gian compile | Vị trí của các thành phần được chỉ định ở run time |
| Thứ tự sắp xếp | Được lưu trữ liên tiếp nhau | Được lưu trữ ngẫu nhiên |
| Cách truy cập | Trực tiếp hoặc ngẫu nhiên | Truy cập một cách tuần tự |
| Chèn và xoá phần tử | Tương đối chậm do cần phải shifting mảng | Dễ dàng, nhanh chóng và tiện lợi |
| Tìm kiếm | Tìm kiếm nhị phân và tìm kiếm tuyến tính | Tìm kiếm tuyến tính |
| Bộ nhớ | Cần ít | Cần nhiều hơn |
| Sử dụng bộ nhớ | Không hiệu quả | Hiệu quả |

**Câu 4: Phân biệt mảng tuyến tính và mảng con trỏ? Cấp phát tĩnh và cấp phát động?**

Mảng tuyến tính: ô nhớ chứa giá trị

Mảng con trỏ: Ô nhớ chứa địa chỉ

* 1. **Cấp phát bộ nhớ tĩnh (Static memory allocation)**:
* Xảy ra trên các **biến tĩnh**và **biến toàn cục**.
* Vùng nhớ của các loại biến này được **cấp phát một lần** khi chương trình bắt đầu chạy và vẫn **tồn tại trong suốt thời gian tồn tại của chương trình**.
* Kích thước của biến/mảng phải được biết tại thời điểm biên dịch chương trình.
  1. **Cấp phát bộ nhớ tự động (Automatic memory allocation)**:
* Xảy ra trên các **tham số hàm** và **biến cục bộ**.
* Vùng nhớ của các loại biến này được cấp phát khi chương trình **đi vào khối lệnh** và được giải phóng khi **khối lệnh bị thoát**.
* Kích thước của biến/mảng phải được biết tại thời điểm biên dịch chương trình.

**Sự khác nhau giữa cấp phát động và cấp phát tĩnh :**

|  |  |  |
| --- | --- | --- |
|  | Cấp phát tĩnh (CPT) | Cấp phát động(CPĐ) |
| Khai báo | Ô nhớ được khai báo sẵn, CPT sẽ lưu trữ toàn bộ giá trị của biến và mảng vào bộ nhớ Stack | Ô nhớ được cấp phát theo nhu cầu. CPĐ sẽ lưu trữ toàn bộ vùng nhớ mà a và Arr tham chiếu tới vào bộ nhớ Heap. |
| Kích thước | Bộ nhớ giới hạn, chỉ có thể cấp phát cho các biến có kích thước nhỏ | Dung lượng bộ nhớ lớn, có thể cấp phát cho các biến có kích thước lớn |
| Thời điểm cấp phát | Biên dịch chương trình | Chạy chương trình |
| Thu hồi bộ nhớ | HĐH tự thu hồi | User tự thu hồi |
| Ưu, nhược điểm | Tốn bộ nhớ  Thiếu bộ nhớ  Khai báo đơn giản | Tránh lãng phí bộ nhớ  Không sợ thiếu  Khai báo phức tạp |

**Câu 5: Khi nào dùng chương trình đệ quy, ví dụ?**

Một bài toán mang tính chất đệ quy khi nó có thể được phân rã thành các bài toán nhỏ hơn nhưng mang cùng tính chất với bài toán ban đầu, các bài toán nhỏ lại được phân rã thành các bài toán nhỏ hơn nữa.

Trong đời sống, ta cũng thường xuyên thấy một số hiện tượng đệ quy, ví dụ như hai chiếc gương đặt đối diện nhau, vòng xoắn ốc (trong vòng xoắn ốc có vòng xoắn ốc nhỏ hơn).

**Câu 6: Danh sách 2 liên kết khác gì với danh sách liên kết kép?**

DSLK kép: Xem định nghĩa SGK, **Không có thứ tự**

DSLK 2**: Có 2 đường liên kết, mỗi đường sắp xếp theo 1 thứ tự**.