LÝ THUYẾT TUẦN 1

1. Biến:

- Biến là một ô nhớ hoặc 1 vùng nhớ dùng để chứa dữ liệu trong quá trình thực hiện chương trình.
- Mỗi biến có một kiểu dữ liệu cụ thể, kích thước của biến phụ thuộc vào kiểu dữ liệu.
- Giá trị của biến có thể được thay đổi, tất cả các bài phải khởi tạo trước khi sử dụng
- Quy cách đặt tên: khuyến nghị đặt tên kiểm CamelCase.
- Cách khai báo: <Kiểu dữ liệu> <Tên biến>;
- Có thể vừa khai báo vừa gán giá trị.
- Địa chỉ của biến là địa chỉ của ô nhớ đầu tiên chứa biến đó, toán tử truy cập địa chỉ của biến: &<tên biến>;
- Biến cục bộ là biến được khai báo trong nội bộ một khối lệnh hoặc bên trong một hàm. Các hàm bên ngoài không thể truy cập biến cục bộ.
- Khi biến cục bộ được định nghĩa, giá trị của biến sẽ không tự động được định nghĩa.
- Biến toàn cục được định nghĩa bên ngoài các hàm và thường được định nghĩa ở đầu chương trình, giữ giá trị xuyên suốt chương trình và tất cả các hàm đều có thể sử dụng biến toàn cục.Biến toàn cục sẽ tự động được khởi tạo giá trị.
- Hằng: có giá trị bất biến. Có 2 cách định nghĩa:

+ #define <Tên hằng> <Giá trị>
+ const <Kiểu dữ liệu> <Tên hằng> = <Giá trị>;

- Giá trị của biến là giá trị của vùng nhớ chứa biến đó.

2. Hàm:

- Hàm là một khối lệnh thực hiện một công việc hoàn chỉnh (module), được đặt tên và được gọi thực thi nhiều lần tại nhiều vị trí trong chương trình.

-Hàm còn gọi là chương trình con (subroutine)

- Hàm có thể được gọi từ chương trình chính (hàm main) ,từ 1 hàm khác hoặc gọi lại chính nó (đệ quy).
- Hàm có thể có giá trị trả về hoặc không. Nếu hàm không có giá trị trả về gọi là thủ tục (procedure - kiểu void).
- Một đoạn chương trình có tên, đầu vào và đầu ra.
- Có chức năng giải quyết một số vấn đề chuyên biệt cho chương trình chính.
- Có thể được gọi nhiều lần với các đối số (arguments) khác nhau.
- Hàm có thể tái sử dụng nhiều lần, dễ sửa lỗi và cải tiến mà không ảnh hưởng nhiều đến toàn bộ chương trình.
- Cách định nghĩa:

```
<Kiểu trả về> <Tên hàm> (Tham số)
{
        <Thân hàm>
}
```

- Parameter (tham số hình thức): Là các thông số đầu vào của hàm, xác định khi khai báo hàm.
- Arguments (tham số thực): là các thông số được nạp vào hàm khi gọi hàm.
- Hàm có thể có cả đầu vào lẫn đầu ra, có một trong 2, hoặc không có cả hai.
- Truyền đối số là việc đưa các thông số cho hàm khi hàm được gọi. Có 2 kiểu truyền tham số:
+ Truyền giá trị: tham số chứa bản sao giá trị của đối số. Thay đổi tham số không ảnh hưởng đến đối số.
+ Truyền tham chiếu:
. Áp dụng cho các tham số khi khai báo có dấu & phía sau kiểu dữ liệu.
. Chỉ có thể truyền các đối số là biến (hoặc hằng nếu tham số khai báo là const)
. Các tham số là tham chiếu không được cấp phát vùng nhớ
. Tham số được truyền tham chiếu sẽ trỏ đến cùng địa chỉ vùng nhớ của đối số truyền cho nó
. Tham số sẽ trở thành một ánh xạ đến đối số. Mọi thay đổi lên tham số sẽ thay đổi luôn đối số.

3. Mảng:
- Biểu diễn một dãy các phần tử có cùng kiểu và mỗi phần tử trong mảng biểu diễn 1 giá trị.
- Kích thước mảng được xác định ngay khi khai báo và không thay đổi.
- Một kiểu dữ liệu có cấu trúc do người lập trình định nghĩa.
- Ngôn ngữ lập trình C/C++ luôn chỉ định một khối nhớ liên tục cho một biến kiểu mảng.
- Khai báo: <Kiểu dữ liệu> <Tên mảng> <Số lượng phần tử của mảng>
- Phải xác định số lượng phần tử của mảng khi khai báo.
- Chỉ số mảng (vị trí trong mảng) là một giá trị số nguyên int.
- Chỉ số bắt đầu là 0 và không vượt quá số lượng phần tử tối đa trong mảng.
- Số lượng các chỉ số mảng = số lượng phần tử tối đa trong mảng.
- Truy xuất phần tử mảng thông qua chỉ số <Tên mảng> [<Chỉ số mảng>].
- Các phần tử mảng là 1 dãy liên tục có chỉ số từ 0 đến <Số phần tử > -1.
- Mảng 2 chiều:
+ Khai báo: <Kiểu dữ liệu> <Tên mảng> [<Số dòng>][<Số cột>].
+ Truy xuất: <Tên mảng>[<Chỉ số dòng>][<Chỉ số cột>].

4. Struct:
- Struct là kiểu dữ liệu trong C/C++ cho phép người dùng kết hợp nhiều dữ liệu khác nhau trong một khối dữ liệu.
- Cách khai báo:

```
struct <Tên struct>
{
	<Kiểu dữ liệu 1> <Tên biến 1>;
	<Kiểu dữ liệu 2> <Tên biến 2>;
	…
	<Kiểu dữ liệu n> <Tên biến n>;
};
```

- Cách truy xuất: không thể truy xuất trực tiếp, được truy xuất qua thành phần của của struct qua toán tử '.': <Tên cấu trúc>.<Tên thành phần>

5. Con trỏ:

- Con trỏ là biến lưu trữ địa chỉ của một biến khác.
- Khai báo: <Kiểu dữ liệu> *<Tên con trỏ>;
- int *p; => p là con trỏ, &p là địa chỉ của con trỏ, *p là giá trị của con trỏ (chứa địa chỉ của biến mà nó trỏ tới).
- Không nên sử dụng con trỏ khi chưa được khởi tạo.
- Cú pháp khởi tạo giá trị con trỏ: <type> pointer = new <type>(value): int *arr = new int[5].
- Có thể đặt tên cho kiểu dữ liệu con trỏ (typedef).
- Có thể sử dụng con trỏ như các kiểu dữ liệu thông thường.
- Con trỏ có thể là tham số hoặc kiểu trả về của hàm.