Khoa CNTT – Đại học Giao thông Vận tải

**LỚP BỔ TRỢ KIẾN THỨC SFIT**

**Môn: Kỹ Thuật Lập Trình**

*Phụ Trách: Lỗ Nguyễn Thành Vinh – CNTT5-K61*

**I. Tổng quan chương trình:**

**-** Gồm 9 phần gắn với các kiến thức liên quan đến học phần Tin học đại cương và Lập trình nâng cao, cũng như một số kiến thức bên ngoài, bao gồm:

* **Phần 1** Tổng quan về C.
* **Phần 2** Biến. Nhập xuất dữ liệu.
* **Phần 3** Các cấu trúc điều khiển
* **Phần 4** Đệ quy và Hàm.
* **Phần 5** Cấu trúc dữ liệu mảng.
* **Phần 6** Chuỗi trong C.
* **Phần 7** Làm việc với file.
* **Phần 8** Con trỏ trong C. Cấp phát bộ nhớ động.
* **Phần 9** Kiểu dữ liệu cấu trúc.
* **Phần 10** Danh sách liên kết.

- Thời gian bổ trợ: 24 buổi học. Trong đó:

* **Buổi 1**: Trình bày phần 1 và 2. Buổi 2 là bài tập tổng kết 2 phần.
* **Buổi 3 và 4**: Trình bày phần 3, bao gồm lý thuyết và bài tập.
* **Buổi 5, 6, 7**: Trình bày phần 4, bao gồm lý thuyết và bài tập.
* **Buổi 8**: Trình bày phần 5, bao gồm lý thuyết và bài tập.
* **Buổi 9**: Bài tập tổng hợp cho 5 phần ở trên.
* **Buổi 10 và 11**: Trình bày phần 6, bảo gồm lý thuyết và bài tập
* **Buổi 12**: Trình bày phần 7, bao gồm lý thuyết và bài tập.
* **Buổi 13**: Bài tập tổng hợp.
* **Buổi 14, 15, 16**: Trình bày phần 8. Bao gồm lý thuyết và bài tập.
* **Buổi 17, 18**: Trình bày phần 9. Bao gồm lý thuyết và bài tập
* **Buổi 19**: Bài tập tổng hợp.
* **Buổi 20, 21**: Trình bày phần 10. Bao gồm lý thuyết và bài tập.
* **Buổi 22, 23**: Bài tập tổng hợp và giải đáp thắc măc.
* **Buổi 24**: Tổng kết quá trình hướng dẫn.

- Thời gian học một buổi: **2 tiếng**, bắt đầu từ **18h đến 20h tối thứ hai hàng tuần**. Cách thức: **Kết hợp cả online và offline**. Địa điểm: Đối với hình thức offline sẽ học tại **Samsung Lab – A4 ĐH GTVT**.

**Phần 1. Tổng quan về ngôn ngữ lập trình C:**

**1.1. Các khái niệm về lập trình:**

- Chương trình là một tập hợp các câu lệnh nhằm hướng dẫn máy tính thực hiện một tác vụ nào đó.

- Lập trình là việc xây dựng, thiết kế ra các chương trình máy tính, sử dụng các ngôn ngữ lập trình.

- Ngôn ngữ lập trình là các công cụ sử dụng để xây dựng lên chương trình máy tính, được chia làm 3 loại là ngôn ngữ máy, hợp ngữ và ngôn ngữ lập trình bậc cao. Để máy tính có thể hiểu được, chương trình máy tính được viết bằng hợp ngữ và ngôn ngữ lập trình bậc cao phải được dịch sang ngôn ngữ máy.

- C thuộc ngôn ngữ lập trình bậc cao.

**1.2. Lịch sử ngôn ngữ C:**

- C được phát triển bởi Dennis Ritchie vào năm 1972, với mục đích chính là tạo ra một ngôn ngữ lập trình hệ thống để xây dựng các hệ điều hành (Unix,…).

- Cái tên C được sử dụng để biểu thị đây là một ngôn ngữ lập trình mới (sau B – Một ngôn ngữ lập trình khác được Ken Thompson phát triển).

- Ngày nay, phần lớn các hệ điều hành quan trọng đều được viết bởi C, bao gồm cả Windows.

**1.3. Một số đặc điểm của ngôn ngữ C:**

- C là một ngôn ngữ lập trình thủ tục, hỗ trợ lập trình có cấu trúc, phạm vi biến, đệ quy,…

- Các lý do khiến C phổ biến:

*i. Dễ học*

*ii. Là một ngôn ngữ lập trình cấu trúc.*

*iii. Cho ra các chương trình hiệu suất cao.*

*iv. Có thể xử lý các hoạt động mức low-level, rất gần với ngôn ngữ máy.*

*v. Có thể được biên dịch sang nhiều nền tảng. Tuy nhiên C lại độc lập nền tảng, phiên bản nền tảng này không thể chạy ở nền tảng kia.*

- Một số tính năng của C:

* Con trỏ: Cho phép tham chiếu đến một vị trí trong bộ nhớ bằng tên được gán cho nó trong chương trình.
* Cấp phát bộ nhớ động.
* Đệ quy
* Thao tác với bit.
* …

- Một số ví dụ hệ thống, chương trình được viết bằng C:

* Hệ điều hành (UNIX OS, Windows,…)
* Các trình biên dịch, thông dịch
* Text editor.
* Driver.
* Cơ sở dữ liệu.

…

**1.4. Cài đặt môi trường cho C:**

- Một chương trình viết bằng C được soạn thảo và lưu trữ trong một file đuôi “.c”. Trong việc lập trình C ta có thể sử dụng rất nhiều trình soạn thảo, như Notepad, Vim, VS Code, Sublime Text,…

- Để chạy chương trình, ta cần dịch file “.c” sang file thực thi “.exe” thông qua một trình biên dịch, từ đó CPU có thể thực thi. Một trong những trình biên dịch phổ biến nhất chính là GNU C/C++.

- Trong học phần này, chúng ta sẽ sử dụng DevC++ hoặc VS Code để thực hành.

**1.4.1. Cài đặt DevC++:**

- Do DevC++ đã tích hợp sẵn cả Text Editor và Trình biên dịch, nên ta không cần cài thêm trình biên dịch nữa.

**1.4.2. Cài đặt môi trường cho VS Code:**

**1.5. Chương trình đầu tiên:**

**-** Ta sẽ viết một chương trình đầu tiên là in ra “Hello, World!” ra ngoài màn hình. Tất nhiên ta có thể in bất kỳ thứ gì mà ta muốn.

#include <stdio.h>

**int** main()

{

printf(“Hello, World!”);

return 0;

}

**-** Việc in ra một dòng chữ ra màn hình có ý nghĩa rất quan trọng: Đầu tiên giúp kiểm tra xem quá trình cài đặt môi trường lập trình đã ổn định chưa, có phát sinh lỗi nào không; Thứ hai là giúp ta hình dung được cấu trúc của một chương trình C.

- Nhìn chung, một chương trình C sẽ gồm các phần:

***i. Documentation:*** *Đây là phần comment của chương trình và không được biên dịch.*

***ii. Linking:*** *Nạp các thư viện vào chương trình hoặc liên kết đến một chương trình khác.*

***iii. Define:*** *Định nghĩa các giá trị hằng số.*

***iv. Global Declaration:*** *Định nghĩa các biến toàn cục sử dụng trong toàn bộ chương trình.*

***v. Main****: Hàm đầu tiên được gọi để thực thi, chứa toàn bộ các câu lệnh chính được chạy.*

***vi. User Defined Function:*** *Chứa các hàm được người lập trình tạo ra nhằm bổ sung và phục vụ cho chương trình.*

- Sau khi soạn thảo chương trình, để chạy ta cần biên dịch chương trình thành mã máy thông qua một trình biên dịch. Quy trình sẽ như sau:

1. Thực hiện các câu lệnh preprocesser (Nạp thư viện, định nghĩa các giá trị hằng số, loại bỏ các câu comment).
2. Trình biên dịch dịch từng dòng code thành các chỉ dẫn máy. Tất cả các lỗi cú pháp nếu được phát hiện sẽ được báo lên trên command line. Cuối cùng nêu không có lỗi, trình biên dịch tạo một file object “.o” chứa toàn bộ mã máy của chương trình chúng ta.
3. Một trình liên kết được sử dụng để liên kết các object file được tạo ra sau quá trình dịch và tổng hợp thành một file thực thi “.exe” hoàn chỉnh.

**1.6. Kiểu dữ liệu, toán tử và biểu thức**

**1.6.1. Các kiểu dữ liệu trong C:**

- C là một ngôn ngữ dạng *kiểu*. Mỗi biến đều được gán một kiểu xác định giá trị nó có thể biểu diễn, cách dữ liệu được lưu trữ trong bộ nhớ và cách các toán tử có thể biểu diễn được trên nó.

- Trong C, kiểu dữ liệu gồm 3 loại chính: Ký tự (Character), Số nguyên (Integer), Số chấm phẩy động (Số thực – float). Mỗi kiểu lại có một kích thước xác định:

|  |  |  |  |
| --- | --- | --- | --- |
| Tên kiểu | Ký hiệu | Kích thước | Đặc tả |
| Ký tự | char | 0 – 256 (1 byte) | %c |
| Số nguyên | int | -2147483648 – 2147483647 (4 byte) | %d |
|  | unsigned Int | 0 – 4294967296 (4 byte) | %u |
|  | long Long Int | -(2^63) – (2^63)-1 (8 byte) | %lld |
| Số thực | float | 8 byte | %f |
|  | double | 16 byte | %lf |

**Phần 4: Hàm và Đệ quy:**

Để có thể nhập dữ liệu cho biến *dummy*, ta sử dụng hàm **scanf**(“%d”, &*dummy*) trong thư viện stdio.h. Muốn xuất một dòng chữ “Road To C++”, ta sử dụng hàm printf(“Road To C++”). Muốn tính căn bậc hai, ta lại sử dụng hàm sqrt() trong thư viện math.h. Vậy câu hỏi đặt ra là, hàm là gì và sử dụng ra sao?

Để sử dụng hàm, ta chỉ cần gọi tên hàm kèm theo hai dấu ngoặc “()”, truyền đầu vào và bùm! Hàm trả về một kết quả thú vị! Vậy cơ chế của hàm như thế nào? Hoạt động ra sao? Một hàm có thể “chứa” một hàm bên trong, vậy một hàm có thể tự gọi chính nó được không?

**4.1.** **Hàm là gì?**

Hàm là một tập hợp các chỉ dẫn nhằm thực hiện một tác vụ nào đó.

**4.1.1. Vì sao cần hàm:**

1. Tránh những tác vụ lặp đi lặp lại
2. Chia nhỏ chương trình thành các tác vụ tính toán nhỏ hơn, cấu trúc nên chương trình.
3. Cho phép người khác sử dụng hàm đã được xây dựng mà không phải xây dựng từ đầu.
4. Tính trìu tượng: Che dấu biến, toán tử, thuật toán đằng sau. Người sử dụng không cần biết hàm hoạt động như thế nào, chỉ cần biết hàm cho được kết quả gì để sử dụng.

**4.1.2. Khai báo hàm:**

Cú pháp khai báo hàm:

**Kiểu-trả-về** Tên-hàm (<Danh-sách-tham-số>)

{

<Câu lệnh>;

**<return** Kết-quả-trả-về>;

}

Trong đó, hàm có thể có trả về hoặc không có trả về. Nếu hàm không có trả về, kiểu trả về là **void**. Ngược lại kiểu trả về có thể là bất kì kiểu dữ liệu nào. Chú ý nếu hàm không trả về, ta không cần keyword return.

*Ví dụ 4.1: Sử dụng hàm để in ra thông tin của một sinh viên truyền vào họ tên cùng tuổi:*

**void** logInformation (const char \*name, int age)

{

printf(“Student Information: \n”);

printf(“Full name: %s\n”, name);

printf(“Age: %d\n”, age);

}

Trong đó:

- Hàm này chỉ có chức năng in ra thông tin sinh viên, do đó ta không cần trả về thêm dữ liệu gì

- tên hàm là logInformation. Tên hàm cần đặt có ngữ nghĩa, tránh các tên hàm như a(), b(),… Tên hàm cũng nên là một động từ, do hàm thường sử dụng để thực hiện một tác vụ nào đó. Thường thì tên hàm sẽ viết thường chữ cái đầu và viết hoa các chữ cái tiếp theo, nhưng một số quy chuẩn của một số nơi thì thay vì viết hoa ta sử dụng dấu gạch nối “\_”.

- Các biến **name, age** được gọi là các tham số. Chúng là các biến được khởi tạo cùng hàm dùng làm ô chứa cho dữ liệu truyền vào hàm. Danh sách tham số có thể có hoặc không.

- Các câu lệnh trong hàm đều nằm bên trong cặp dấu “{}”.

- Keyword **return** không xuất hiện do hàm không trả về giá trị nào. Tuy nhiên điều đó không có nghĩa là thêm **return** vào hàm sẽ bị lỗi. Đối với kiểu **void**, **return** sẽ mang ý nghĩa kết thúc hàm ngay tại thời điểm gọi **return** (Điều này hữu ích nếu hàm của ta gặp một điều kiện bất lợi cần kết thúc ngay).

*Ví dụ 4.2: Hàm tìm giá trị nhỏ nhất của một mảng:*

**int** getMin(**int** arr[], **int** n)

{

**int** min = arr[0];

**for** (**int** i = 1; i < n; i++)

{

**if** (min >= arr[i])

min = arr[i];

}

**return** min;

}

Hàm này duyệt toàn bộ mảng và so sánh các phần tử của mảng với một giá trị mốc (min). Nếu giá trị này lớn hơn một phần tử nào thì cập nhật lại giá trị gốc. Khi đó, để có thể tìm giá trị nhỏ nhất của một mảng, ta chỉ cần gọi hàm getMin() và truyền vào các giá trị cần thiết, hàm sẽ cho ta giá trị thoả mãn:

**int** main()

{

**int** arr[] = {1, 3, -2, 5, 8, 9, -9, 0, -1};

**int** n = 9;

**int** minOfArray = getMin(arr, n);

**printf**(“Minimize of array: **%d\n**”, minOfArray);

}

Kết quả in ra sẽ là:

-9

Ta cũng có thể truyền trực tiếp hàm như một *đối số (1):*

**printf**(“Minimize of array: **%d\n**”, getMin(arr, n);

**Chú ý:** Hai biến arr và n trong hàm main() hoàn toàn khác hai biến arr và n trong hàm getMin(). Ta sẽ tìm hiểu sự khác nhau này sâu trong bài học.

(1): Khái niệm tham số và đối số sẽ được trình bày kỹ hơn trong bài. Ta có thể hiểu đối số là giá trị truyền vào hàm.

**4.1.3. Return**

Return là từ khoá chỉ định giá trị trả về của hàm có trả về giá trị.

Cú pháp:

**return** *biểu-thức*

Trong đó, biểu thức có thể là một giá trị cụ thể:

**int** isOdd(**unsign int** n)

{

**if** (n % 2 == 0)

**return** 0;

**return** 1;

}

Hoặc một biến như hàm getMin() ở trên, hoặc là một biểu thức:

**int** getLengthOfLine(**float** xA, **float** yA, **float** xB, **float** yB)

{

**return** (sqrt((xA – xB) \* (xA – xB) + (yA – yB) \* (yA – yB));

}

**Chú ý:** Biểu thức sau return sẽ được tự động ép kiểu sang kiểu trả về.

Hoàn toàn có thể không có một biểu thức đằng sau return. Khi đó, hàm gặp return sẽ tự động huỷ:

*Ví dụ 4.3: Hàm sắp xếp tăng dần một mảng arr:*

**void** bubbleSort(**int** arr[], **int** n)

{

**if** (n <= 1)

**return;**

**int** temp;

**for (int** i = 0; i < n; i++)

{

**for** (**int j =**

Như vậy nếu trong trường hợp người dùng truyền vào hai số a, b nhỏ hơn hoặc bằng 0, hàm sẽ tự động bị huỷ và trả về kết quả rác.

**4.1.4. Tham số (Parameters) và đối số (Arguments):**

Ta xét ví dụ 4.2:

**1**

**int** getMin(**int** arr[], **int** n)

{

// Do something

}

**int** main()

{

**int** arr[] = {1, 3, -2, 5, 8, 9, -9, 0, -1};

**int** n = 9;

**2**

**int** minOfArray = getMin(arr, n);

**printf**(“Minimize of array: **%d\n**”, minOfArray);

}

Hai cặp biến arr và n ở dòng **1** và **2** có gì khác nhau?

Ở dòng **1**, cặp biến arr và n là hai biến được khởi tạo cùng với hàm khi hàm được gọi, sử dụng như một cổng truyền dữ liệu vào hàm. Chúng được gọi là các *tham số (Parameters)*. Còn ở dòng **2**, cặp biến arr và n là hai biến được khởi tạo bên ngoài hàm, chúng chứa dữ liệu để nạp vào hàm và được truyền vào hàm. Do đó chúng được gọi là các *đối số (Arguments).*

Như vậy:

**Tham số** là các biến được khởi tạo cùng hàm, có vai trò như các biến nhận dữ liệu cho hàm.

**Đối số** là các biến chứa dữ liệu và được truyền vào hàm thông qua tham số.

Tham số giống như các ô chứa tạm thời còn đối số là các giá trị thực tế. Một biến chỉ được gọi là đối số khi nó được truyền vào trong hàm.

Ta có thể minh hoạ tham số và đối số thông qua sơ đồ sau:

Graphical user interface, application, Teams

Description automatically generated

**4.2. Phạm vi và vòng đời của biến:**

**4.2.1. Phạm vi của biến:**

Một biến chỉ được hoạt động trong phạm vi mà nó được khai báo.

Xét một ví dụ về một chương trình sau:

*Ví dụ 4.4. Hàm atoi chuyển đổi một chuỗi trở thành một số nguyên:*

**int** atoi(**const char** *s*[])

{

**int** i, result = 0, sign;

**for** (i = 0; isspace(*s*[i]); i++)

        ; // Skip white space.

    sign = (*s*[i] == '-') ? -1 : 1;

**if** (*s*[i] == '-' || *s*[i] == '+')

    {

        i++; // Skip this sign

    }

**while** (*s*[i] != '**\0**')

    {

**if** (isdigit(*s*[i]))

        {

            result = 10 \* result + (*s*[i] - '0');

        }

        i++;

    }

**return** result \* sign;

}

**4.3. Cách hoạt động của hàm trong C:**

**4.4. Hàm có nhiều tham số.**

**4.5. Truyền giá trị (Call by values) và truyền tham chiếu (Call by references)**

**4.6. Khai báo hàm (Function Decralation):**