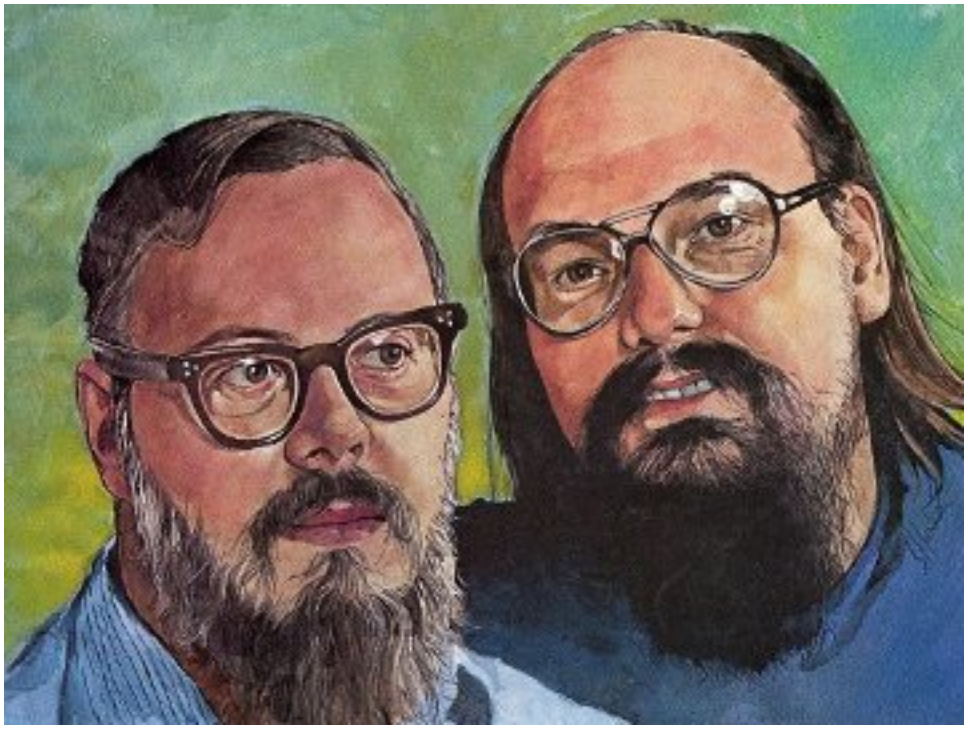
Table of Contents

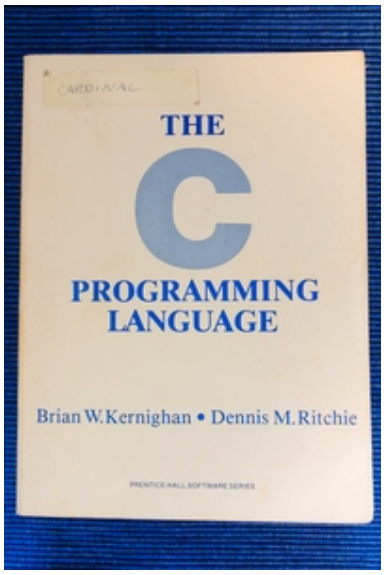
1. Introduction

1. History

- 1970, Dennis Ritchie và Brian W.Kernighan bắt đầu phát triển ngôn ngữ lập trình C tại Bell Labs của tập đoàn AT&T. Ngôn ngữ C dựa trên nhiều nền tảng của dựa trên ngôn ngữ lập trình BCPL và B.



- 1978, Dennis Ritchie và Brian W.Kernighan giới thiệu về ngôn ngữ C với cộng đồng qua tựa sách “The C Programming Language ”. Trong khoảng thời gian này, ngôn ngữ C vẫn được tiếp tục phát triển và nhận được rất nhiều sự đóng góp của cộng đồng và tổ chức. Điều này dẫn tới việc nhiều hiện thực và tính năng của ngôn ngữ C không thống nhất trên các hệ thống/compiler (trình biên dịch).



- Ban đầu, ngôn ngữ C được thiết kế để hoạt động trên hệ điều hành UNIX, với chiếc máy

DEC-PDP-11 của Dennis. Sau này, ngôn ngữ C được phát triển để có thể hoạt động trên nhiều hệ máy và hệ điều hành khác nhau.
- 1983-1989, viện tiêu chuẩn của hoa kì (ANSI) thành lập một ủy ban và đưa ra định nghĩa và các chuẩn mực rõ ràng về ngôn ngữ C nhằm thống nhất lại sự không đồng nhất do các tổ chức hiện thực compiler thời bấy giờ. Kết quả là chúng ta có chuẩn ANSI C như bây giờ.
- 1990, tiêu chuẩn ISO được phát hành trên khắp thế giới với một vài điều chỉnh nhỏ dựa trên tiêu chuẩn ANSI C.
- Từ đó đến nay, có nhiều phiên bản tiêu chuẩn được ban hành nữa, nhưng đáng chú ý nhất là tiêu chuẩn ISO/IEC 9899:1999. Tiêu chuẩn mới nhất được phát hành là ISO/IEC 9899:2011, các trình biên dịch hiện nay chỉ hỗ trợ một phần của chuẩn này.

2. Introduction
   - C là một ngôn ngữ lập trình đa mục đích (General Programming Language), tức là nó có thể dùng cho nhiều mục đích khác nhau. Thông thường, ngôn ngữ lập trình C được sử dụng trong embedded development (lập trình nhúng), system development (lập trình hệ thống) và software development (lập trình phần mềm). Ngoài ra, ngôn ngữ lập trình C còn có thể sử dụng trong lập trình Web thông qua cgi/fastcgi . Nhiều chương trình dạy lập trình viên trên thế giới sử dụng C như một ngôn ngữ nhập môn, bởi vì thông qua C, người học hiểu cặn kẽ về những khái niệm của hệ thống, quản lý bộ nhớ, và là nền tảng cho họ trong các ngôn ngữ khác như C++, Java, Objective-C.
3. Install environment

2. First Example and Explain

1. HelloWorld Example

- Cách hay nhất để bắt đầu một việc đó bắt tay vào làm nó, tôi đã luôn nghĩ như vậy. Vì thế, thay vì đưa ra cho các bạn một mớ lý thuyết lằng nhằng, tôi mong muốn các bạn hãy tự tay mình làm một ứng dụng nho nhỏ trước và chúng ta cùng phân tích nó. Bạn sẽ dễ tiếp thu hơn đấy, dù sao cũng tự bạn tạo ra một ứng dụng nho nhỏ mà.

- Để khởi động, bạn hãy dùng text editor (phần mềm soạn thảo văn bản) bất kỳ để viết ra những đoạn code đầu tiên nhé. Nếu bạn đang sử dụng Window, bạn có thể xài Notepad, Notepad++ hoặc Sublime Editor; nhưng nếu bạn đang xài Linux, bạn có thể dùng Vi, Vim, Emacs, Gedit. Trong ví dụ này, mình sẽ save file của mình với tên là "hello.c" với nội dung

```
#include <stdio.h>
int main() {
    // Print message hello to screen
    printf("hello world!\n");
    return 0;
}
```

- Sau khi bạn đã lưu code vào file, bạn hãy dùng c compiler để để build file. Dùng commandline, di chuyển tới thư mục bạn đã lưu file hello.c, gõ đoạn command như sau:

```
gcc -o hello.exe hello.c
```

- Kết quả là bạn sẽ được 1 file tên là "hello.exe" ngay thư mục bạn vừa gõ lệnh. Chạy thử file đó và tận hưởng thành quả nào:

```
hello world!
```

2. Explain

- Giờ chúng ta hãy phân tích đoạn code do chính tay bạn viết nhé. Dòng đầu tiên trong đoạn mã bạn vừa viết được gọi là Preprocessor Derective

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

- Chúng ta sẽ có dịp nói kỹ hơn về khái niệm này trong những bài viết sau. Hiện tại bạn có thể hiểu dòng code này sẽ load thư viện stdio, và bạn có thể sử dụng những chức năng có sẵn trong thư viện stdio. Nói thêm cho bạn biến stdio là từ viết tắt là Standard Input/Output, nên chúng ta có thể sử dụng những chức năng liên quan đến việc nhập xuất dữ liệu.

- Tiếp theo là một phần rất, rất là quan trọng trong một ứng dụng C:

```
int main() {
    ...
    ...
}
```

- Cấu trúc trên dùng để khai báo một function (hàm) tên là main, chúng ta cũng sẽ đi vào chi tiết về function sau. Bạn có thể hiểu đây là start point, nơi mà ứng dụng C sẽ chạy những dòng lệnh đầu tiên trong này. Function main() này rất là quan trọng, vì nếu không có nó, ứng dụng C sẽ không biết bắt đầu chạy từ đâu và đương nhiên là khi bạn compile với gcc, bạn sẽ nhận được lỗi là thiếu function main().

- Tiếp theo, chúng ta hãy đi vào trong hàm main và xem có gì bên trong đó nhé.

```
// Print message hello to screen
```

Đoạn code có ký tự "//" phía đầu được gọi là comments (chú thích). Có thể hiểu đây là những đoạn chú thích của bạn, dành cho những người khác đọc đoạn code của bạn. Thường thì comment rất hữu ích nếu như đoạn code của bạn rắc rối hoặc bạn đơn giản muốn nhắn nhủ gì đó cho những người sau đọc đoạn code của bạn. Lưu ý là comments sẽ không được compile (biên dịch) và bị loại bỏ khi bạn chạy chương trình.

Comment bằng "//" chỉ hỗ trợ bạn viết ghi chú trên 1 dòng. C cung cấp một cách comment khác trên nhiều dòng với cú pháp như sau:

```
/*
    Print message hello to screen
    Second line of comments
*/
```

Nội dung ở giữa ký hiệu "/*" và "*/" sẽ được C hiểu là ghi chú và được bỏ qua khi biên dịch.

```
printf("hello world!\n");
```

- Tiếp theo đó là function printf(), một function của library stdio. Ta có thể sử dụng function printf() của library stdio là do đã **"#include <stdio.h>"**. Function này có khả năng xuất ra màn hình đoạn chữ mà ta truyền vào. Trong ví dụ của chúng ta là đoạn chữ **"hello world!\n"**, ta sẽ thấy kết quả là chúng ta thấy được đoạn chữ "hello world!" được xuất ra màn hình.

- Vậy ký tự "\n" có nghĩa là gì, bạn có thắc mắc về điều đó không ? Trong những ngôn ngữ lập trình, những ký tự đó gọi chung là special escape characters. Có nhiều special escape characters khác nhau và "\n" là một trong số đó, nó có nghĩa là xuống hàng. Khi bạn xuất ra ký tự "\n", nó sẽ xuống hàng. Bạn thử thêm vài kỹ tự "\n" vào đoạn chữ và compile lại xem nào.

- Cuối cùng là dòng code:

```
return 0;
```

- "return" dùng để thông báo kết thúc một function và sẽ thoát ra khỏi function đó ngay lập tức. Trong hàm main của chúng ta, bạn có thể thấy "return 0", đó có nghĩa là thoát khỏi hàm main() và thông báo cho người gọi ứng dụng biết phần mềm chạy ổn. Nếu bạn "return 1" thì sẽ thông báo ứng dụng chạy có lỗi. Để hiểu rõ hơn về return, chúng ta sẽ quay lại sau trong những chương tiếp theo nói về function.

- Vậy là bạn đã hiểu tất cả những gì bạn viết ra rồi đấy ! Thích không nào.

3. Thử thách

- Từ bài này trở đi, sau mỗi chương, mình sẽ gửi các bạn vài thử thách, để giúp bạn hiểu rõ hơn về toàn bộ chương ấy. Free for dicussing nhé!!

- Bài tập kỳ này:

> Viết một ứng dụng có thể in ra đoạn chữ sau đây:
>
> Hello world!
> I am a C programmer, and I love you.
>
> I can input 2 breaklines.

- Thử nhé!

3. Variable

1. Variable Concept

– Vào thời điểm khởi nguyên của máy tính, các lập trình viên phải viết ra những chương trình của họ bằng mã nhị phân. Họ phải thao tác trên những vùng nhớ máy tính thông qua những con số địa chỉ, đấy thật sự là một việc hết sức kinh khủng và khó khăn.

– May thay, ngôn ngữ lập trình C hỗ trợ cho các bạn những định danh, cho phép các bạn truy xuất những vùng nhớ trên máy tính thông qua những cái tên có nghĩa hơn. Bạn cảm thấy một vùng nhớ có tên là **"numberOfChildren"** có nghĩa hơn hay 291048 có nghĩa hơn nào :) Và C gọi chúng là Variable, cung cấp cho lập trình viên cách thức giao tiếp với vùng nhớ dễ dàng hơn, tập trung nhiều vào logic của ứng dụng hơn là phải cố nhớ xem những dãy số loằng ngoằng có nghĩa là gì.

– Khởi động 1 tí nhé, chúng ta hãy chỉnh sửa ứng dụng "hello world" của kì trước và làm nó trở nên thú vị hơn với variable nhé

```
#include <stdio.h>
int main() {
    int myAge = 25;
    // print message to screen
    printf("hello %d!\n", myAge);
    return 0;
}
```

– Bạn còn nhớ cách để build một ứng dụng và chạy nó mà đúng không, thử và kiểm tra kết quả nào.

```
Hello 25!
```

– Thú vị không nào, giờ chúng ta sẽ đi sơ qua những gì bạn mới chỉnh sửa nhé. Đầu tiên, bạn đã thêm vào 1 dòng trong function main() như sau:

```
int myAge = 25;
```

– Dòng này có nghĩa nghĩa là gì, bạn thử đoán xem... Tạo một vùng nhớ, có tên là **myAge**, có kiểu dữ liệu là **int**, và có chứa giá trị là 25. Về kiểu dữ liệu, bạn sẽ được giới thiệu ngay thôi, nhưng bây giờ, mình có thể chúc mừng bạn là bạn đã biết cách tạo ra một variable rồi đấy. Tiếp theo bạn đã làm gì với variable tên là myAge nhé:

```
printf("hello %d!\n", myAge);
```

– Bạn đã chỉnh sửa lại thông tin truyền vào function **printf** một tí, thay vì là "hello world", chúng ta đang đang có "hello %d", và bạn truyền thêm vào **myAge** ở cuối function printf. Lúc này, bạn cứ tạm hiểu là %d đóng vai trò như một người giữ chỗ, khi ứng dụng chạy, nó sẽ giá trị của

variable **myAge** thay thế vào vị trí **%d**, và kết quả là màn hình hiện lên ""hello 25!"

- Vậy là xong, 80% thông tin về variable, bạn đã nắm rồi đó, giờ chúng ta đi vào chi tiết từng phần của 20% còn lại thôi.

2. Variable Name

```
int myAge = 25;
```

– Qua ví dụ ở trên, bạn có thể tạo ra variable với một cái tên do bạn đặt (phần được tô đen), vậy thì tên của variable có phải theo nguyên tắc gì không ? Chắc chắn là phải có rồi, những nguyên tắc bạn cần phải tuân theo khi đặt tên cho variable là:

- Ký tự đầu tiên của tên phải là ký tự (a-z) hoặc là dấu gạch dưới (_)
- Những ký tự tiếp theo có thể là ký tự, dấu gạch dưới (_) hoặc là ký số (0-9)
- Không được đặt tên trùng với những keywords có sẵn của C:

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

– Một điều cần lưu ý nữa là có phân biệt chữ hoa chữ thương trong ngôn ngữ C, vì vậy nên **MyAge** và **myAge** hoàn toàn khác nhau nhé các bạn.

3. Variable Type and Sizes

Việc lưu trữ và quản lý dữ liệu trong máy tính thường sẽ có 2 thông tin quan trọng cần chú ý, một là dữ liệu được lưu tại ví trí nào trên thanh bộ nhớ, hai là nó đang lưu cái gì.

```
int myAge = 25;
```

Với variable name, chúng ta đã có được thông tin đầu tiên. Và variable type (kiểu dữ liệu - phần được tô đen) sẽ quyết định cái gì được lưu trong vùng nhớ đó, độ lớn bao nhiêu và dữ liệu đó dạng gì.

Ngôn ngữ lập trình C cung cấp nhiều kiểu dữ liệu cho ta lựa chọn để sử dụng, trong bài này, chúng ta sẽ quan tâm đến 3 nhóm dữ liệu chính là Integer, Floating point number và void

1. Nhóm Integers
   - Kiểu dữ liệu int:

   Kiểu dữ liệu **int** cho phép người dùng lưu trữ số nguyên có giá trị từ -32768 đến 32767 (tương đương 2 bytes). Ví dụ: 535, 10, 7, -100

   Trong một số máy tính hoặc thiết bị, kiểu int có thể chứa tới 4 bytes dữ liệu, tức là từ

-2,147,483,648 2,147,483,647

Ngôn ngữ C còn hỗ trợ những kiểu đặc biệt nhằm hỗ trợ cho kiểu int:

Kiểu dữ liệu	Kích thước chứa	Giá trị cho phép
unsigned int	2 hoặc 4 bytes	0 đến 65535 hoặc 0 đến 4294967295
short int	2 bytes	-32768 đến 32767
unsigned short int	2 bytes	0 đến 65535
long	4 bytes	-2147483648 đến 2147483647
unsigned long	4 bytes	0 đến 4,294,967,295

2. Nhóm Floating point Number
   - Kiểu dữ liệu float

   Kiểu dữ liệu float cho phép người dùng lưu trữ dạng dữ liệu kiểu số thực, hay còn gọi là số có dấu chấm động. Vd: 10.5, 73.1111.

   Kiểu dữ liệu float hỗ trợ độ dài là 4 bytes và đảm bảo độ chính xác trong vòng 6 số thập phân. Vì vậy, hãy cẩn thận khi so sánh hoặc tính toán khi mà độ dài phần thập phân nhiều hơn 6 nhé

   - Kiểu dữ liêu double

   Cũng như kiểu dữ liệu float, kiểu double dùng để lưu trữ dữ liệu dạng số thực nhưng với độ lớn và độ chính xác cao hơn. Kiểu double hỗ trợ lên tới 8 bytes độ lớn và đảm bảo độ chính xác lên tới 15 ở phần thập phân.

   Ngoài ra, ngôn ngữ lập trình C còn hỗ trợ một kiểu đặc biệt nữa là long double, hỗ trợ lên đến 10 bytes độ lớn và độ chính xác ở phần thập phân lên tới 19.

   Thật ra ở một giới hạn công việc nhất định, mình chưa gặp phải trường hợp phải xài tới kiểu dữ liệu long double để lưu trữ... Mong là một ngày nào đó có cơ hội xài

Kiểu dữ liệu	Kích thước chứa	Giá trị cho phép	Độ chính xác
float	4 bytes	1.2E-38 đến 3.4E+38	6 số thập phân
double	8 bytes	2.3E-308 đến 1.7E+308	15 số thập phân
long double	10 bytes	3.4E-4932 đến 1.1E+4932	19 số thập phân

3. Kiểu dữ liệu Void

   Kiểu dữ liệu void là một kiểu dữ liệu đặc biệt, kiểu dữ liệu này thể hiện rằng không có gì được lưu trữ và độ lớn của nó bằng 0. Do là kiểu dữ liệu đặc biệt nên cách xài nó cũng đặc biệt hơn so với bình thường. Chúng ta sẽ nhắc về nó sau trong những bài tới. Tạm thời thì các bạn cứ biết có một kiểu dữ liệu tên là void và không lưu trữ bất kì thứ gì cả.

4. Constants

Constant có thể được hiểu là những variable nhưng chỉ có thể đọc (readonly) và không thể sửa nội dung. Tất cả các số, ký tự hay chuỗi ký tự trong C đều được gọi là constants. Ví dụ: 123, "hello world", 'c', ... Trong ví dụ các bạn viết, phần constants chính là phần được tô đen:

```
int myAge = 25;
```

Chúng ta có thể đặt tên cho variable constants bằng cách dùng keyword const

```
const int myConstantAge = 25
int myAge = myConstantAge;
```

C hỗ trợ cho chúng ta nhiều loại constants, chúng ta sẽ đi qua những loại chính như sau.

- int, như ví dụ bạn vừa viết. Constant dạng int là một dãy số: 12345.
- long int constant được viết dưới dạng thêm chữ l hoặc L vào cuối giá trị số: 12345L hoặc 12345l
- float constant được viết dưới dạng thêm chữ f hoặc F vào cuối giá trị: 12f hoặc 12F
- double constant thường được viết dưới dạng có dấu chấm động (1.5) hoặc dạng 15e-1
- long double constants được viết giống như double constants nhưng thêm l hoặc L phía cuối: 1.5l hoặc 1.5L
- char constants là một số nguyên, được viết dưới dạng 1 ký tự nằm giữa dấu ngoặc đơn: 'a', 'b', 'c'. Giá trị lưu dữ dưới bộ nhớ là dạng số nguyên và tham chiếu với bảng mã ký tự ASCII char constants character còn hỗ trợ những ký tự đặt biệt được gọi là escape characters:

Escape character	Giải thích
\a	alert (bell) character
\b	backspace
\f	formfeed
\n	newline
\r	carriage return
\t	horizontal tab
\v	vertical tab
\\	backslash
\?	question mark
\'	single quote
\"	double quote
\ooo	octal number
\xhh	hexadecimal number

- Các kiểu constants còn hỗ trợ cách viết theo dạng bát phân (octa): 051, 023, 083 hoặc thập lục phân (hexa): 0x84, 0x48, 0x69

5. Declarations

   Bất kì variable nào cũng phải được khai báo trước khi sử dụng. Cú pháp để khai báo variable trong C như sao:

```
//variableType variableName;
int variable1;
unsigned int  variable2;
short int  variable3;
unsigned short int  variable4;
long int variable5;
unsigned long int variable6;
float variable7;
double variable8;
long double variable9;
char variable10;
```

Ngoài ra, C còn cung cấp một phương thức để vừa khai báo variable, vừa gán giá trị cho variable.

```
//variableType variableName;
int variable1 = -10;
unsigned int  variable2 = 15;
short int  variable3 = -20;
unsigned short int  variable4 = 20;
long int variable5 = -100001L;
unsigned long int variable6 = 987653L;
float variable7 = 3.5f;
double variable8 = 2e10;
long double variable9 = 5.5;
char variable10 = 'c';
```

6. Printf

Phần cuối cùng của chương này, mình muốn nói tới cách sử dụng đơn giản variable trong ứng dụng của bạn thông qua function printf. Printf là một function được cung cấp bởi thư viện **stdio**, có chức năng xuất ra màn hình nội dung với những định dạng linh hoạt. Một trong những định dạng mà mình đã giới thiệu với bạn là **%d**, để hiểu rõ hơn về những định dạng này, chúng ta hãy xem qua nhé.

Trong C, những định dạng này được gọi tên là format specifiers, dùng để thay thế giá trị vào với quy ước nhất định. Một format specifier có cấu trúc như sau:

```
%[flags][width][.precision][length]specifier
```

Chúng ta hãy đi phân tích từng thành phần

1. Specifier

specifier	Kết quả nhận được	Example
d *or* i	Kiểu dữ liệu số nguyên	392
u	Kiểu dữ lệu số nguyên dương	7235
o	Kiểu dữ liệu số nguyên dương theo hệ bát phân	610
x	Kiểu dữ liệu số nguyên dương theo hệ thập lục phân	7fa
X	Kiểu dữ liệu số nguyên dương theo hệ thập lục phân (viết hoa)	7FA
f	Kiểu dữ liệu số thực	392.65
F	Kiểu dữ liệu số thực (viết hoa)	392.65
e	Kiểu dữ liệu số thực với ký tự e	3.9265e+2
E	Kiểu dữ liệu số thực với ký tự E (viết hoa)	3.9265E+2
g	Sử dụng: %e hoặc %f	392.65
G	Sử dụng: %E hoặc %F	392.65
a	Kiểu dữ liệu số thực theo hệ thập lục phân (viết thường)	-0xc.90fep-2
A	Kiểu dữ liệu số thực theo hệ thập lục phân (viết hoa)	-0XC.90FEP-2
c	ký tự	a
s	chuỗi ký tự	sample
%	hiện ra ký hiệu % nếu chúng ta viết %%	%

Trăm lời vàng ngọc không bằng 1 cọc ví dụ:

```
#include<stdio.h>

int main() {
    //variableType variableName;
    int variable1 = -10;
    unsigned int  variable2 = 15;
    float variable7 = 3.5f;
    double variable8 = 2e10;
    long double variable9 = 5.5;
    char variable10 = 'c';

    printf("print value of %%d and %%i: %d, %i\n", variable1, variable2);
    printf("print value of %%u: %u, %u\n", variable1, variable2);
    printf("print value of %%o, %%x, %%X: %o, %x, %X\n", variable1, variable1, variable1);
    printf("print value of %%f, %%F, %%e, %%E, %%g and %%G: %f, %F, %e, %E, %g, %G\n", variable7, variable7, variable7, variable7, variable7, variable8);
    printf("print value of %%c, %%s: %c, %s\n", variable10, "message");
}
```

2. Flags

flags	**Ghi chú**
-	Nếu có định nghĩa độ dài cần hiển thị (mục tiếp theo, width), nếu ta thêm dấu -, sẽ canh trái nội dung hiện ra. Mặc định là canh phải.
+	Bắt buột giá trị số hiển thị ký hiệu + cho số dương và - cho số âm.
(space)	Độn thêm ký tự khoảng cách vào bên trái nếu như giá trị hiện ra không đủ độ dài theo quy định
#	Đối với specifiers kiểu số phức (a, A, e, E, f, F, g, G), tự động xuất ra phần thập phân. Đối với specifiers kiểu o, x, X thì hiện ký tự 0, 0x, 0X
0	Độn thêm ký tự khoảng cách vào bên trái nếu như giá trị hiện ra không đủ độ dài theo quy định

Trăm lời vàng ngọc không bằng 1 cọc ví dụ:

```
#include <stdio.h>

int main() {
    int variable1 = -10;
    unsigned int  variable2 = 15;
    int variable11 = 0x7;

    printf("Value of %%10d: %10d\n", 1);
    printf("Value of %%-10d: %-10d\n", 2);
    printf("Value of %%+d and %%+d: %+d and %+d\n", variable1, variable2);
    printf("Value of %% d and %% d: % d and % d\n", variable1, variable2);
    printf("Value of %%#g and %%#x: %#g and %#x\n", variable2, variable11);
}
```

3. Width

width	Ghi chú
number	Số lượng ký tự ít nhất phải được in ra. Nếu như giá trị thực ngắn hơn giá trị width này thì sẽ mặc định độn thêm bên trái ký tự khoảng trắng cho đủ. Nếu giá trị thực dài hơn giá trị width này thì sẽ hiện bình thường và không bị cắt bớt.

```
#include <stdio.h>

int main() {
    int variable1 = 10;
    printf("Value of %%5d: %5d\n", variable1);
}
```

4. Precision

precision	Ghi chú
. number	Nếu như đang định dạng kiểu specifiers của số nguyên (d, i, o, u, x, X): số lượng ký số nhỏ nhất phải được ghi ra ở phần thập phân. Nếu phần thập phân ngắn hơn giá trị của precision này, thì sẽ add thêm số 0. Nếu phần thập phân dài hơn giá trị thức của precision, phần thập phân sẽ hiện ra đầy đủ Nếu như specifiers là a, A, e, E, f, F: In ra số lượng phần thập phân sẽ hiện ra sau dấu . động

```
#include <stdio.h>


int main() {
    float variable1 = 10.5f;
    float variable2 = 10.12345f;
    printf("Value of %%.5f and %%.3g: %.5f and %.3g\n", variable1, variable2);
}
```

7. Bài tập về nhìn, à nhầm, về nhà
   Vận dụng kiến thức về variable và printf, viết một ví dụ dùng để in tất cả các giá trị mà bạn biết của mỗi loại variable type

4. Operators

Operators, còn được gọi là toán tử, là những cú pháp, ký tự cho phép lập trình viên có thể thực hiện những hành động tính toán hoặc kiểm tra trên variables. Trong ngôn ngữ C, chúng ta chia ra làm 7 loại operations khác nhau, và hôm nay chúng ta sẽ tìm hiểu về chúng.

1. Arithmetic (số học)

   Là những toán tử liên quan đến việc tính toán số học. Căn bản nhất là những phép tính: cộng (+), trừ (-), nhân (*) và chia (\). Cách sử dụng, bạn có thể xem qua ví dụ bên dưới:

```
#include <stdio.h>
int main() {
    printf("5 + 8 = %d\n", 5 + 8);
    printf("9 - 2 = %d\n", 9 - 2);
    printf("10 / 2 = %d\n", 10 / 2);
    printf("7 * 8 = %d\n", 7 * 8);
}
```

   Ngoài 4 phép tính căn bản trên, C còn hỗ trợ "%" dùng để lấy phần dư của một phép chia

```
#include <stdio.h>
int main() {
    printf("9 % 3 = %d\n", 9 % 3);
    printf("10 % 3 = %d\n", 10 % 3);
}
```

   Ngôn ngữ C còn cung cấp một phép tính "++" và "--" dùng để tăng hoặc giảm một đơn vị. Chúng được dùng với 2 cách với ý nghĩa khác nhau

```
number++ hoặc ++number
```

   Nếu "++" nằm phía trước của number, number sẽ tăng 1 đơn vị trước khi thực hiện các phép tính khác.

   Nếu "++" nằm phía sau của number, number sẽ tăng 1 đơn vị sau khi thực hiện các phép tính khác.

   Để hiểu rõ hơn về cách sử dụng khác nhau của ++ và --, chúng ta hãy đến với ví dụ sau đây:

```
#include <stdio.h>

int main() {
  int number;
  number = 5;
  int result1 = number++;
  number = 5;
  int result2 = ++number;
  printf("Value of number++ is %d\n", result1 );
  printf("Value of ++number is %d\n", result2 );
}
```

Kết quả nhận được là:

```
Value of number++ is 5
Value of ++number is 6
```

Như các bạn thấy, việc đặt toán tử ++ hoặc -- phía trước hoặc phía sau của variable sẽ cho ra kết quả và ý nghĩa hoàn toàn khác nhau. Với number++, giá trị của number sẽ được tăng sau khi giá trị của number được gán cho result1. Với ++number, giá trị của number sẽ được tăng trước khi number được gán cho result2.

Chúng ta có thể hiểu như sau:

```
int result1 = number++;
```

Sẽ tương đương với 2 bước sau:

```
int result1 = number;
number = number + 1;
```

và

```
int result1 = ++number;
```

Sẽ tương đương với 2 bước sau:

```
number = number + 1;
int result1 = number;
```

Tương tự với phép toán -- nhé.

2. Relational

Relational operators là những toán tử dùng để so sánh 2 giá trị với nhau.

Operator	Chú Thích
==	Kiểm tra 2 giá trị bằng nhau. Nếu bằng nhau, kết quả trả về là 1, nếu không bằng nhau, kết quả trả về là 0.
!=	Kiểm tra 2 giá trị không bằng nhau. Nếu bằng nhau, kết quả trả về là 0, nếu không bằng nhau, kết quả trả về là 1.
>	Kiểm tra lớn hơn. Nếu thỏa điều kiện, kết quả trả về là 1, nếu không thỏa điều kiện, kết quả trả về là 0.
<	Kiểm tra bé hơn. Nếu thỏa điều kiện, kết quả trả về là 1, nếu không thỏa điều kiện, kết quả trả về là 0.
>=	Kiểm tra lớn hơn hoặc bằng. Nếu thỏa điều kiện, kết quả trả về là 1, nếu không thỏa điều kiện, kết quả trả về là 0.
<=	Kiểm tra bé hơn hoặc bằng. Nếu thỏa điều kiện, kết quả trả về là 1, nếu không thỏa điều kiện, kết quả trả về là 0.

Chúng ta sẽ đến với ví dụ

```
#include <stdio.h>


int main() {
    int number1 = 10;
    int number2 = 5;
    int number3 = 10;


    printf("Value of number1 == number2: %d\n", number1 == number2);
    printf("Value of number1 == number3: %d\n", number1 == number3);
    printf("Value of number1 == number3: %d\n", number1 == number3);
    printf("Value of number1 != number2: %d\n", number1 != number2);
    printf("Value of number1 != number3: %d\n", number1 != number3);
    printf("Value of number1 > number2: %d\n", number1 > number2);
    printf("Value of number1 > number3: %d\n", number1 > number3);
    printf("Value of number2 < number3: %d\n", number2 < number3);
    printf("Value of number3 < number2: %d\n", number3 < number2);
    printf("Value of number1 >= number2: %d\n", number1 >= number2);
    printf("Value of number1 >= number3: %d\n", number1 >= number3);
    printf("Value of number2 <= number3: %d\n", number2 <= number3);
    printf("Value of number1 <= number3: %d\n", number1 <= number3);
}
```

3. Logical

   Logical operators là những toán tử "and", "or", hoặc "not" dùng để kiểm tra tính đúng sai của nhiều biểu thức. Thường được dùng kết hợp với relational operators

Operator	Chú thích	Ví dụ
&&	Thường được gọi là toán tử AND. Nếu cả hai vế (bên trái và bên phải của toán tử AND) đều không phải chứa giá trị 0, thì kết quả là 1. Ngược lại, giá trị nhận được sẽ có kết quả là 0.	(1 > 0) && (10 < 12) Có thể phát biểu như sau: Nếu 1 > 0 VÀ 10 < 12 thì thỏa điều kiện.
||	Thường được gọi là toán tử OR. Nếu cả hai vế (bên trái và bên phải của toán tử OR) có một giá trị khác 0, thì kết quả là 1. Ngược lại, giá trị nhận được sẽ có kết quả là 0.	(1 > 0) || (10 > 12) Có thể phát biểu như sau: Nếu 1 > 0 HOẶC 10 > 12 thì thỏa điều kiện.
!	Thường được gọi là toán tử NOT, nó sẽ trả kết quả ngược lại mà kết quả mà nó xử lý.	Thông thường: (1 > 0) trả kết quả là 1 Nhưng nếu xài NOT: ! (1 > 0) kết quả là 0 ! (1 < 0) kết quả là 1

```
#include <stdio.h>


int main() {
    printf("Value of (1 > 0) && (10 < 12): %d\n", ((1 > 0) && (10 < 12)));
    printf("Value of (1 > 0) || (10 > 12): %d\n", ((1 > 0) || (10 > 12)));
    printf("Value of !((1 > 0) && (10 < 12)): %d\n", !(((1 > 0) && (10 < 12))));
    printf("Value of !((1 > 0) || (10 > 12)): %d\n", !(((1 > 0) || (10 > 12))));
    return 0;
}
```

   Để có thể hiểu rõ hơn về logical operators, chúng ta hãy đến với ví dụ sau:

4. Bitwise

   Bitwise là những operators dùng để thao tác trên từng bit dữ liệu. Như các bạn đã biết, hệ thống máy tính hiện nay thường được lưu trữ và hoạt động bản chất dựa trên từng bit dữ liệu (sẽ chứa giá trị 1 hoặc 0). C cung cấp bitwise như một cách thức để lập trình viên có thể thao tác trên bits dữ liệu ấy.

Operator	Chú thích	Ví dụ
&	Toán tử AND của bitwise, sẽ so sánh từng bits giữa 2 mẫu cho trước và cho ra kết quả trên tiêu chí: + Nếu cả 2 cùng có giá trị 1, kết quả sẽ là 1 + Ngược lại, kết quả sẽ là 0	Giả sử: A = 1000 0001 B = 0000 1001 Kết quả

		A & B = 0000 0001
|	Toán tử OR của bitwise, sẽ so sánh từng bits giữa 2 mẫu cho trước và cho ra kết quả trên tiêu chí: + Chỉ cần 1 trong 2 mẫu có giá trị là 1, kết quả lấy là 1 + Nếu cả 2 cùng có giá trị là 0, kết quả sẽ là 0	Giả sử: A = 1000 0001 B = 0000 1001 Kết quả A | B = 1000 1001
^	Toán tử Exclusive OR của bitwise, sẽ so sánh từng bits giữa 2 mẫu cho trước và cho ra kết quả trên tiêu chí: + Nếu cả 2 mẫu cùng có giá trị 1, kết quả sẽ là 0 + Nếu cả 2 mẫu cùng có giá trị 0, kết quả sẽ là 0 + Nếu một trong 2 mẫu có giá trị 1, cái còn lại có giá trị 0, kết quả sẽ là 1	Giả sử: A = 1000 0001 B = 0000 1001 Kết quả A ^ B = 1000 1000
~	Toán tử đảo bit, sẽ đảo bits dựa trên nguyên tắc: + Nếu bit là 0, đảo thành 1 + Nếu bit là 1, đảo thành 0	Giả sử: A = 1000 0001 Kết quả ~A = 0111 0111
<<	Dịch bits dữ liệu sang bên trái	Giả sử: A = **1000 0001** Kết quả A >> 2 = 00**10 0000**
>>	Dịch bits dữ liệu sang bên phải	Giả sử: A << 2 = **0000 01**00

5. Assignment

Toán tử gán giá trị được bạn sử dụng từ đầu tới giờ. Bạn thử dùng vài giây để nhớ lại xem bạn dùng nó khi nào nhé.

```
A = B
```

Assignment operators là ký tự "=" dùng để gán giá trị từ bên phải của dấu "=" (A) qua cho bên tay trái của dấu "=" (B). Ví dụ:

```
#include <stdio.h>
int main() {
   int myAge = 25;
   // print message to screen
   printf("hello %d!\n", myAge);
   return 0;
}
```

Trong ví dụ trên, bạn gán giá trị 25 vào trong variable có tên là myAge. Assignment operators còn hỗ trợ giá giá trị của một variable này sang variable khác:

```
#include <stdio.h>
int main() {
    int myAge = 25;
    int hisAge = myAge;
    // print message to screen
    printf("hello %d!\n", myAge);
    return 0;
}
```

Assignment operators cũng hỗ trợ cách viết như sau:

```
#include <stdio.h>
int main() {
    int hisAge = myAge = 25;
    // print message to screen
    printf("hello %d!\n", myAge);
    return 0;
}
```

Lưu ý là cách hoạt động của assignment operator luôn luôn từ phải qua trái.

6. Type Casting Operators

   Type casting là những cách thức cho phép chuyển đổi giữa các variable type (kiểu giữ liệu) với nhau. Để đổi variable type, chúng ta sử dụng cú pháp sau

```
(type_name) expression
```

   Ví dụ:

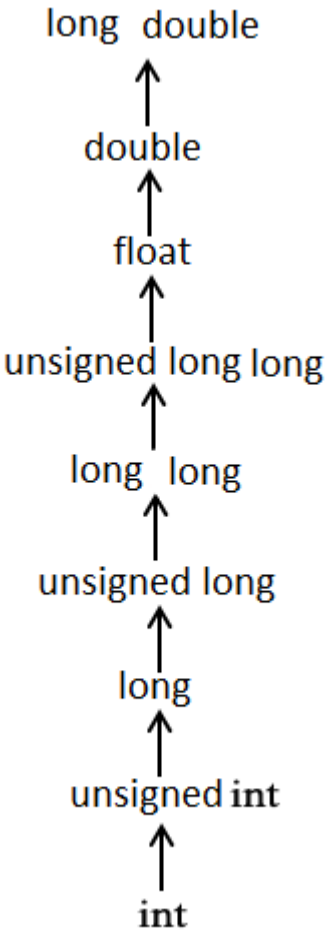
```
#include <stdio.h>
int main() {
    int numberOfCandies = 10;
    int totalAmount = 25;
    double price = (double) 25 / 10;
    printf("Price of a candy: %f\n",
price);
    return 0;
}
```

   Ngôn ngữ C còn cung cấp một cách thức type casting tự động nữa, được gọi là Integer

Promotion. Đây là một quy trình tự động đổi những variable type nhỏ hơn int hoặc unsigned int thành kiểu int hoặc unsigned int để sử dụng. Ví dụ:

```
#include <stdio.h>
int main() {
    int integer = 10;
    char character = 'c'; // ascii of c is 99
    int sum = integer + character;
    printf("%d\n", sum);
    return 0;
}
```

Một cơ chế tự động thực hiện type casting nữa bạn cần biết đó là Usual Arithmetic Conversion, cơ chế này sẽ tự động chuyển variable type theo nguyên tắc từ dưới lên trên của hình sau:



```
#include <stdio.h>
int main() {
    int number1 = 10;
    int number2 = 20;
    float sum = number1 + number2;
    printf("%f\n", sum);
    return 0;
}
```

Ví dụ trên cho ta thấy number1 và number2 là kiểu integer, giá trị từ phép + của number1 và number2 cũng là kiểu integer, nhưng nó tự động đổi varible type để phù hợp với kiểu float.

7. Operator precedence

   Khi các operators được sử dụng chung với nhau. Một số expression sẽ có quyền ưu tiên và sẽ được thực thi trước những operators khác, ví dụ như trong ví dụ sau đây:

   ```
   10 + 5 * 2
   ```

   Phép nhân sẽ được ưu tiên thực hiện trước phép tính cộng. Bảng liệt kê dưới đây sẽ mô tả cho chúng ta biết thứ tự ưu tiên đó:

Độ ưu tiên	Các toán tử	Chú thích
Nhóm 1 (độ ưu tiên cao nhất)	++	++ được đặt sau, vd: number++
	--	-- được đặt sau, vd: number--
	++	++ được đặt trước, vd: ++number
	--	++ được đặt trước, vd: ++number
Nhóm 2	!	Not
	~	Đảo bits
	(*variable type*)	Chuyển variable type, vd: (float) 10
	*	Nhân
Nhóm 3	/	Chia
	%	Chia lấy phần dư
Nhóm 4	+	Cộng
	-	Trừ
Nhóm 5	<<	Dịch bits sang trái
	>>	Dịch bits sang phải
	<	Bé hơn
Nhóm 6	<=	Bé hơn hoặc bằng
	>	Lớn hơn
	>=	Lớn hơn hoặc bằng
Nhóm 7	==	So sánh bằng
	!=	So sánh không bằng
Nhóm 8	&	Bitwise AND

Nhóm 9	^	Bitwise XOR (exclusive or)
Nhóm 10	|	Bitwise OR
Nhóm 11	&&	Logical AND
Nhóm 12	||	Logical OR
Nhóm 13 (độ ưu tiên thấp nhất)	=	Direct assignment

5. Expressions & Statements and Blocks

1. Expression

   Expression là một cú pháp được cấu tạo gồm những variables, operators và method invocations (sẽ tìm hiểu sau). Expression thường trả ra một kết quả duy nhất và kiểu dữ liệu sẽ tùy thuộc vào variables, operators và method invocations. Những ví dụ được in đậm dưới đây sẽ làm bạn hiểu rõ hơn về expression:

```
int number1 = 10;
```

```
number1 = 5 + 6 / 2;
```

```
printf("%d", 10 * 3);
```

```
int result = myAge > yourAge;
```

2. Statements

   Statement là câu lệnh xử lý hoàn chỉnh và thường được kết thúc bằng dấu ";".

```
#include <stdio.h>
int main() {
    int number1 = 10;
    int number2 = 20;
    float sum = number1 + number2;
    printf("%f\n", sum);
    return 0;
}
```

   Có một sốt statement đặc biệt không cần dấu ";" để kết thúc, ví dụ như control flow statement. Chúng ta sẽ tìm hiểu ở chương tiếp theo về control flow statement.

3. Blocks

   Block là một tập hợp nhiều statements lại với nhau. Thường được mở đầu bằng dấu "{" và kết thúc bằng dấu "}". Một block quen thuộc mà chúng ta sử dụng từ đầu tới giờ đó là block của function main()

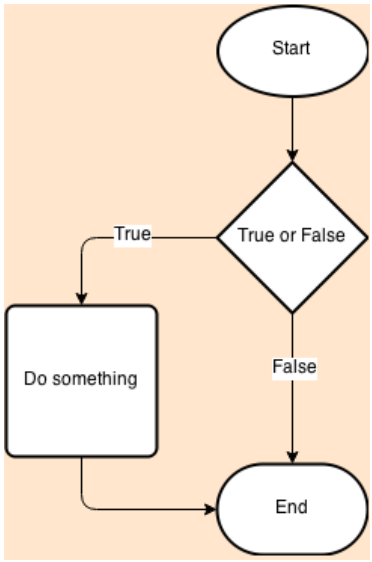
```
#include <stdio.h>
int main() {
    int number1 = 10;
    int number2 = 20;
    float sum = number1 + number2;
    printf("%f\n", sum);
    return 0;
}
```

6. Control flows

Từ đầu tới giờ, các bạn đã được giới thiệu về cách viết một ứng dụng. Nhưng cách viết đấy chỉ chạy từ trên xuống dưới, thực thi hết dòng này tới dòng khác, một cách tuần tự. Để có thể giải quyết những tính huống phức tạp hơn, C cung cấp một vài cách thức để điều khiển luồng đi của ứng theo nhiều cách khác chứ không chỉ tuần tự từ trên xuống dưới.

Trong chương này, mình sẽ giới thiệu với các bạn những cách dùng thức điều khiển luồng chảy của ứng dụng như mong muốn

1. If



Trong cuộc sống hằng ngày, ta thường có hành vi thực hiện một hành động nào đó dựa trên một điều kiện nhất định. Ví dụ như nếu nếu trời đang mưa, bạn sẽ mang theo chiếc dù để đi ra ngoài; nếu trời không mưa, bạn sẽ không mang theo dù. C cũng cung cấp cho ta một cách thức tương tự để quyết định có hay không thực hiện một đoạn code nhất định.

Và cú pháp để làm việc đó trong C như sau:

```
If (condition)
   expression
```

Khi chương trình chạy và thấy cú pháp If, nó sẽ kiểm tra điều kiện trong *condition* trước, nếu kết quả là true, chương trình sẽ thực thi expression, nếu kết quả là false, chương trình sẽ bỏ qua và không thực thi expression.

Trong C, true được đại diện bằng giá trị khác 0 (thường là 1), và false được đại diện bởi những giá trị bằng 0.

Để hiểu rõ hơn về if, ta đến với ví dụ như sau:

```
#include <stdio.h>

int main() {
   int myMoney = 5;
   int yourMoney = 10;

   printf("My money is %d$\n", myMoney);
   if(myMoney > 7)
      printf("I have money to buy Icecream");

   printf("Your money is %d$\n", yourMoney);
   if(yourMoney > 7) {
      printf("You have money to buy Icecream\n");
   }
}
```

Kết quả ta có được như sau:

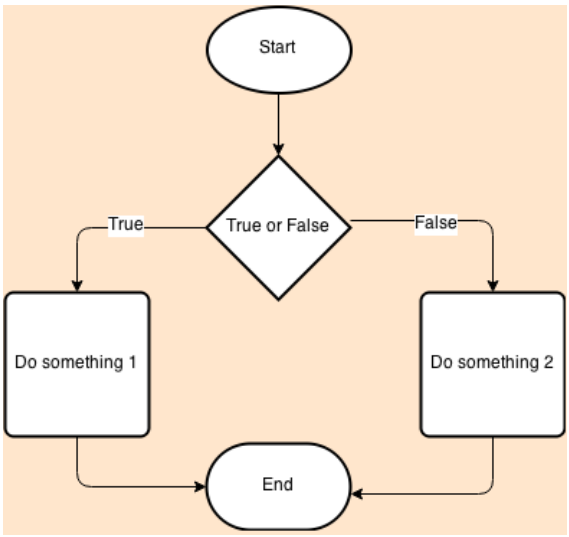
My money is 5
Your money is 10
You have money to buy Icecream

Ta có thể thấy ***(myMoney > 7)*** không thỏa điều kiện nên ***printf("I have money to buy Icecream");*** không được thực thi.*Trong khi đó* ***(yourMoney > 7)*** thỏa điều kiện nên ***printf("You have money to buy Icecream\n");*** được thực thi.

Cú pháp If còn cho phép ta thực thi một block thay vì expression nhờ { và }

```
if(yourMoney > 7) {
      printf("You have money to buy Icecream\n");
      printf("And can I borrow you %d$\n", (yourMoney - 7));
}
```

2. If … else ...



Được xem là cú pháp mở rộng của If, If ... else cho phép người dùng linh hoạt hơn trong việc thực thi với điều kiện không thỏa yêu cầu. Trong thực tế, If...else có thể hiểu như phát biểu sau: nếu trời đang mưa, bạn sẽ mang theo chiếc dù để đi ra ngoài; nếu trời không mưa, bạn sẽ mang theo chiếc nón kết.

```
if (condition)
    expression1
else
   expression2
```

Cú pháp If ... Else sẽ kiểm tra điều kiện trong *contidion*, nếu thỏa điều kiện là true, chương trình sẽ thực thi expression1, nếu điều kiện là false, chương trình sẽ thực thi expression2. Để hiểu rõ hơn, ta lấy làm qua ví dụ.

```
#include <stdio.h>

int main() {
   int myMoney = 5;

   printf("My money is %d$\n", myMoney);
   if(myMoney > 7)
      printf("I have money to buy Icecream\n");
   else
      printf("I don't have money to buy Icecream\n");
}
```

Ta có kết quả:

```
if(myMoney > 7) {
    printf("I have money to buy Icecream\n");
    printf("I will smile\n");
} else {
    printf("I don't have money to buy Icecream\n");
    printf("I can't smile\n");
}
```

Kết quả ta có là:

My money is 5$
I don't have money to buy Icecream

Trong ví dụ, do ***(myMoney > 7)*** không thỏa điều kiện nên vế ***else*** sẽ được thực thi, và ta có kết quả ***I don't have money to buy Icecream*** .

Cú pháp If ... else cũng cho phép ta thực thi block thay vì expression

```
if(myMoney > 7) {
    printf("I have money to buy Icecream\n");
    printf("I will smile\n");
} else {
    printf("I don't have money to buy Icecream\n");
    printf("I can't smile\n");
}
```

C còn cung cấp một cách thức gần giống cú pháp if...else nhưng chỉ cần viết trên một dòng như sau:

```
condition ? expression1 : expression2
```

Ứng dụng sẽ kiểm tra điều kiện condition, nếu thỏa điều kiện thì expression1 sẽ được thực thi, nếu không thỏa điều kiện, expression2 sẽ được thực thi.

Một điều đặc biệt nữa là cú pháp trên được xem là một expression nên sẽ có giá trị trả về để ta gán nó vào biến khác hoặc method để sử dụng.

Để hiểu rõ hơn, chúng ta đến với ví dụ sau:

```
#include <stdio.h>

int main() {
   int myMoney = 5;
   (myMoney > 7) ? printf("I smile\n") : printf("I can't smile\n");

   int yourMoney = 3;
   myMoney = (yourMoney > 0) ? (myMoney + yourMoney) : myMoney;
   printf("My money at this time: %d\n", myMoney);
}
```

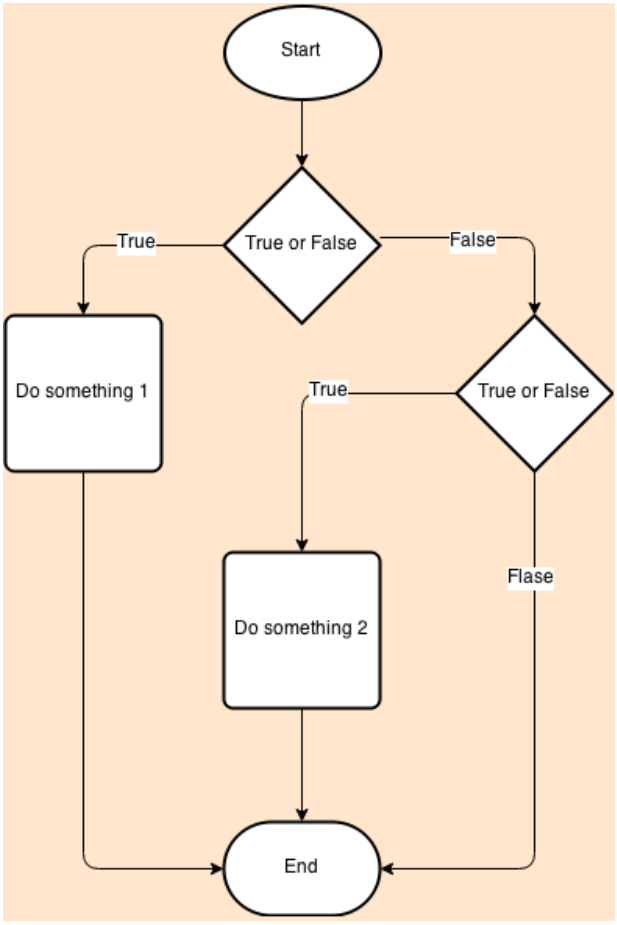
Kết quả ta có là:

I don't have money to buy Icecream

My money at this time: 8

Ta có thể thấy, ***(yourMoney > 0) ? (myMoney + yourMoney) : myMoney*** có thể dùng để gán giá trị lại vào variable ***myMoney*** như trên ví dụ trên hoặc sử dụng để gọi hàm printf như khi ta sử dụng if...else...

3. If … else if ...



Cũng là phần mở rộng của cấu trúc if, cấu trúc if ... else ... if ... cho phép đưa thêm nhiều điều kiện trong trường hợp điều kiện trước đó không thỏa mãn. Cú pháp này giống việc sử dụng if liên tục nhau.

```
if(condition1)
   expression1
else if(condition2)
   expression2
```

Cũng như cú pháp if, khi ứng dụng gặp cú pháp if ... else if ..., ứng dụng sẽ kiểm tra điều kiện của condition1, nếu condition1 thỏa mãn, expression1 sẽ được thực thi; nếu condition1 không thỏa mãn, ứng dụng sẽ tiếp tục kiểm tra tiếp condition2. Nếu như có bất kỳ condition nào thỏa mãn điều kiện, ứng dụng thoát ra khỏi cú pháp if...else if sau khi thực thi expression mà không kiểm tra các condition tiếp theo.

Ta có thể tiếp tục nối thêm vào else hoặc else if nếu cần xử lý nếu như condition2 không thỏa

```
if(condition1)
   expression1
else if(condition2)
   expression2
else
   expression3
```

```
if(condition1)
   expression1
else if(condition2)
   expression2
else if(condition3)
   expression3
```

Để hiểu rõ hơn, ta đến vói ví dụ như sau

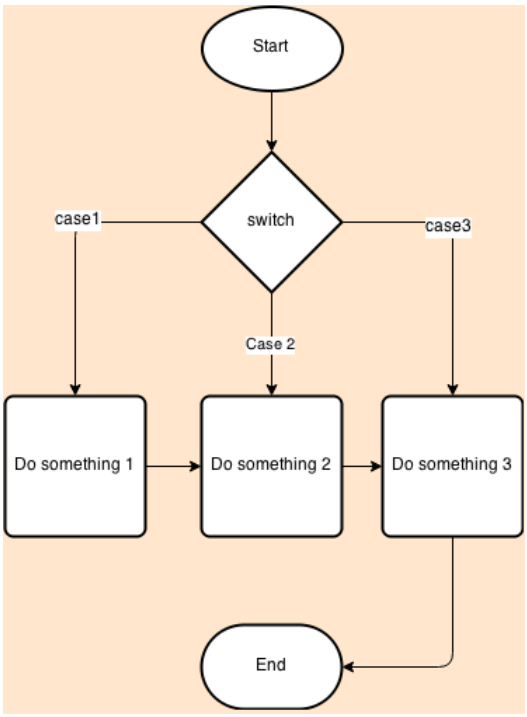
```
#include <stdio.h>

int main() {
   float myMark = 5.5;

   printf("My mark is %.1f\n", myMark);

   if(myMark > 9) {
      printf("I'm perfect\n");
   } else if(myMark > 7) {
      printf("I'm great\n");
   } else if(myMark > 5) {
      printf("I'm cool\n");
   } else if(myMark > 3) {
      printf("I think I'm cool\n");
   } else if(myMark > 1) {
      printf("Maybe I'm cool\n");
   } else {
      printf("Alright, alright\n");
   }
}
```

Kết quả:

My mark is 5.5

I'm cool

4. Switch



Một cách thức khác để thể hiện sự lựa chọn bằng code thay vì sử dụng if, đó là switch. Switch được viết với cú pháp như sau:

```
switch(expression) {
   case const-express: statements
   case const-express: statements
   ...
   default: statements
}
```

Khi sử dụng swich, ứng dụng sẽ kiểm tra xem expression và const-express của từng case có bằng nhau hay không. Nếu có bất kỳ const-express thỏa điều kiện (bằng với expression), thì statements tương ứng của case đó sẽ được thực thi. Sau đó, ứng dụng sẽ thực thi tiếp các statements của case tiếp theo mà không cần kiểm tra const-express nữa.

Trong trường hợp ứng dụng không tìm thấy bất kì case nào thỏa điều kiện (const-express thỏa điều kiện), statements của default sẽ được thực thi. Ngoài ra, default có thể không cần được định nghĩa trong cú pháp switch. lúc đó cú pháp sẽ trông giống như vầy:

```
switch(expression) {
   case const-express: statements
   case const-express: statements
}
```

Chú ý:

+ const-express là những variable constants được nhắc tới ở chương variable.

+ Các const-express trong một switch phải khác nhau. Nếu có bất kỳ const-express bị trùng, bạn sẽ không thể build được ứng dụng.

Để hiểu rõ hơn, ta đến với ví dụ:

```
#include <stdio.h>

int main() {
   int myMark = 9;



   printf("My mark is %d\n", myMark);

   switch(myMark) {
      case 10:
         printf("My rewards include a PSP\n");
      case 9:
         printf("My rewards include a bicycle\n");
      case 8:
         printf("My rewards include a T-shirt\n");
      default:
         printf("My rewards include a IceCream\n");
   }
}
```

Và kết quả là:

My mark is 9

My rewards include a bicycle

My rewards include a T-shirt

My rewards include a IceCream

Như bạn thấy, ứng dụng sẽ so sánh ***myMark*** với từng case, và case 9 thỏa điều kiện nên statement của case 9 và các statements của các case phía đưới đều được thực thi lần lượt.

Nhưng trong thực tế, không phải lúc nào bạn cũng muốn tất cả các statements phía dưới được thực thi, bạn chỉ muốn riêng statement của case thỏa điều kiện được thực thi thôi. Và break làm việc đó một cách xuất sắc. Break statement có tác dụng thoát khỏi cấu trúc swtich ngay lập tức. Có nghĩa sau khi thực thi xong statement, ta đặt break để thoát khỏi switch.

Sửa ứng dụng lại một chút và ta có đoạn code như sau:

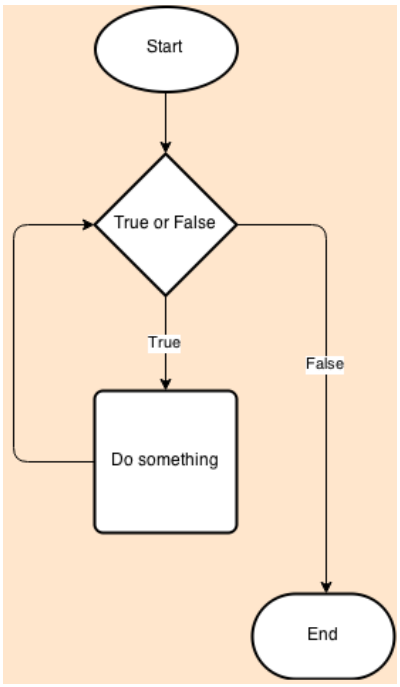
```
#include <stdio.h>

int main() {
   int myMark = 9;



   printf("My mark is %d\n", myMark);

   switch(myMark) {
      case 10:
         printf("My rewards is a PSP\n");
         break;
      case 9:
         printf("My rewards is a bicycle\n");
         break;
      case 8:
         printf("My rewards is a T-shirt\n");
         break;
      default:
         printf("My rewards is a IceCream\n");
   }
}
```

Kết quả ta có được là:

My mark is 9
My rewards is a bicycle

5. Vòng lặp while

Đến thời điểm này, bạn đã có đủ kha khá kiến thức để có thể viết ứng dụng rồi. Nhưng bạn nghĩ sao nếu ai đó nhờ bạn viết một ứng dụng xuất ra màn hình 1000 dòng "I love you". Viết thì cũng được đó, nhưng sẽ tốn kha khá thời gian của bạn đấy :) Ngôn ngữ C biết được khó khăn đó đó nên đã cung cấp một cách thức cho phép bạn lặp đi lặp lại một hành động. Cấu trúc ấy được gọi là vòng lặp.



Vòng lặp đầu tiên giới thiệu cho bạn đó là vòng lặp while, vòng lặp while sẽ kiểm tra điều kiện, nếu thỏa điều kiên (giá trị là true) thì sẽ thực hiện hành động, sau đó quay lại kiểm tra điều kiện tiếp. Vòng lặp while chỉ ngừng khi điều kiện không thỏa (giá trị là false).

```
while(condition)
        expression
```

Cấu trúc while sẽ kiểm tra điều kiện condition, nếu condition có giá trị true, expression sẽ được thực hiện, rồi sau đó condition được kiểm tra tiếp. Nếu condition có giá trị false, vòng lặp bị ngừng lại. Để hiểu rõ hơn, ta đến ví dụ sau đây:

```
#include <stdio.h>


int main() {
   int counter = 1;
   while(counter < 7)
      printf("Number of counter: %d\n", counter++);
}
```

Kết quả ta có:

Number of counter: 1
Number of counter: 2
Number of counter: 3
Number of counter: 4
Number of counter: 5
Number of counter: 6

Khi Counter lớn hơn 7, vòng lặp sẽ bị ngưng lại và không thực hiện được. Vòng lặp C cũng cho phép lặp lại cả block code:

```
#include <stdio.h>

int main() {
   int counter = 1;
   while(counter < 7) {
      printf("Number of counter: %d\n", counter);
      counter++;
   }
}
```

6. Vòng lặp for

   Cũng giống như vòng lặp while, nhưng vòng lặp for cho phép ta khai báo việc khởi tạo biến đếm, kiểm tra, và tính toán biến đếm trên cùng một dòng

```
for(expression1; expression2; expression3)
        expression4
```

   Nếu biểu diễn vòng biến đổi vòng lặp for thành while, ta có được như sau.

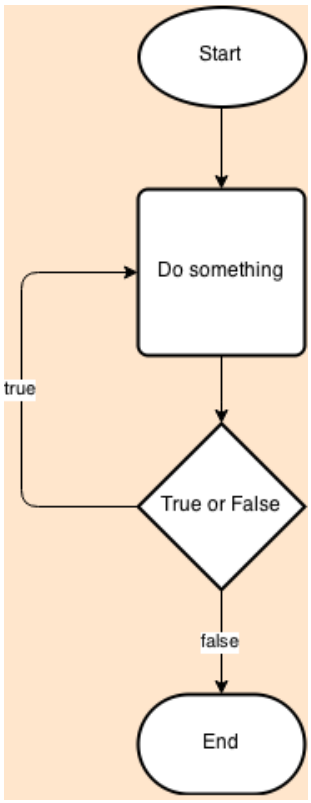
```
expression1
while(expression2;) {
        expression4
        expression3
}
```

   Dùng cùng ví dụ của vòng lặp while, ta có thể sử dụng vòng lặp for như sau:

```
#include <stdio.h>

int main() {
   int counter;
   for(counter = 1; counter < 7; counter++) {
      printf("Number of counter: %d\n", counter);
   }
}
```

   Trong thực tế, người ta thường sử dụng vòng lặp for để duyệt giá trị từ x->y. Trong ví dụ trên, ta kiểm duyệt qua các giá trị từ 1 tới 7.

7. Vòng lặp do … while ...



Vòng lặp do...while có cú pháp như sau:

```
do {
    expression
} while();
```

Vòng lặp do ... while hoạt động gần giống với vòng lặp while, nhưng khác một chỗ là vòng lặp while kiểm tra condition trước, rồi mới quyết định có tiếp tục thực hiện expression hay không. Trong khi đó, vòng lặp do...while thực hiện expression trước rồi mới kiểm tra điều kiện condition xem có tiệp tục vòng lặp hay không. Để hiểu rõ hơn, ta đến với ví dụ sau:

```
#include <stdio.h>

int main() {
    int counter = 0;
    do {
        counter++;
        printf("Number of counter: %d\n", counter);
    } while(counter < 7);
}
```

Kết quả ta có:

Number of counter: 1
Number of counter: 2
Number of counter: 3
Number of counter: 4

Number of counter: 5
Number of counter: 6
Number of counter: 7

8. break and continue

Một số trường hợp, chúng ta cần thoát khỏi vòng lặp for/while/do..while hoặc cấu trúc switch ngay lập tức. Ngôn ngữ C cung cấp chúng ta statement break để làm việc đó. Khi gặp statement break, ứng dụng sẽ ngay lập tức thoát khỏi for/while/do...while và switch ngay lập tức mà không cần xét tới điều kiện.

```
#include <stdio.h>


int main() {
   int counter;
   for(counter = 1; counter < 7; counter++) {
      if(counter == 3) {
         printf("We call break and you won't see counter = 3 or higher\n");
         break;
      }

      printf("Number of counter: %d\n", counter);
   }
}
```

Và kết quả:

Number of counter: 1
Number of counter: 2
We call break and you won't see counter = 3 or higher

Bên cạnh break, C còn có thêm statement continue. Continue dùng để ngừng xử lý tại thời điểm đó và đi kiểm tra điều kiện ngay lập tức. Continue được sử dụng cho while/for và do...while.

```
#include <stdio.h>

int main() {
   int counter;
   for(counter = 1; counter < 7; counter++) {
      if(counter == 3) {
         printf("We call continue and you won't see counter = 3\n");
         continue;
      }

      printf("Number of counter: %d\n", counter);
   }
}
```

Kết quả:

Number of counter: 1
Number of counter: 2
We call continue and you won't see counter = 3
Number of counter: 4
Number of counter: 5
Number of counter: 6

9. goto

Trong ngôn ngữ C, ta có thể đánh dấu đánh dấu một điểm trong code với một cái tên, sau đó tại bất cứ đâu, ta có thể gọi dùng goto để đi đến điểm đánh dấu ấy.

Trong thực tế, goto ít khi được sử dụng vì lý do khó kiểm soát và làm code của bạn trở nên rối loạn. Trong phần hướng dẫn, mình sẽ cho ví dụ để các bạn có thể hình dung cách hoạt động của nó, nhưng ở các bài sau, mình sẽ không sử dụng nó thường xuyên.

```
#include <stdio.h>

int main() {
   int counter;
   for(counter = 1; counter < 7; counter++) {
      if(counter == 10) {
         die:
         printf("Application will be die\n");
         break;
      }

      printf("Number of counter: %d\n", counter);
      goto die;
   }
}
```

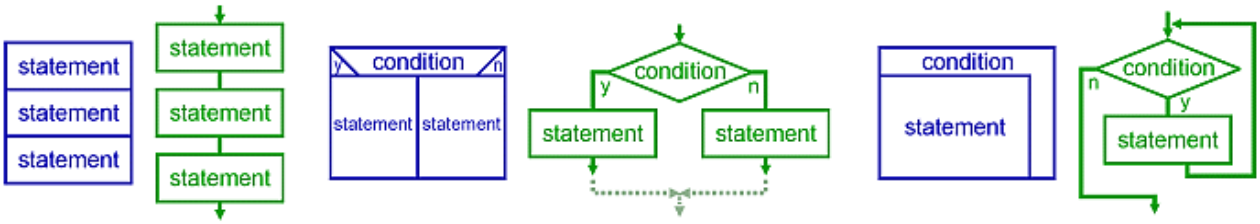
Và ta có kết quả:

Number of counter: 1
Application will be die

7. Functions

1. Structured Programming (lập trình có cấu trúc hay lặp trình hướng cấu trúc)

   Structured Programming là phương pháp lập trình nhằm cải thiện tính trong sáng, chất lượng của code và giảm thiểu thời gian phát triển ứng dụng. Thật chất, structured programming giải quyết những mã rối được sinh ra từ hướng lập trình bằng lệnh goto. Lập trình hướng cấu trúc cố gắng làm cho cấu trúc của trương trình phản ánh những gì mà chương trình đó có thể thực hiện được thông qua những kỹ thuật phân chia module, component, block code, vòng lặp for và while.

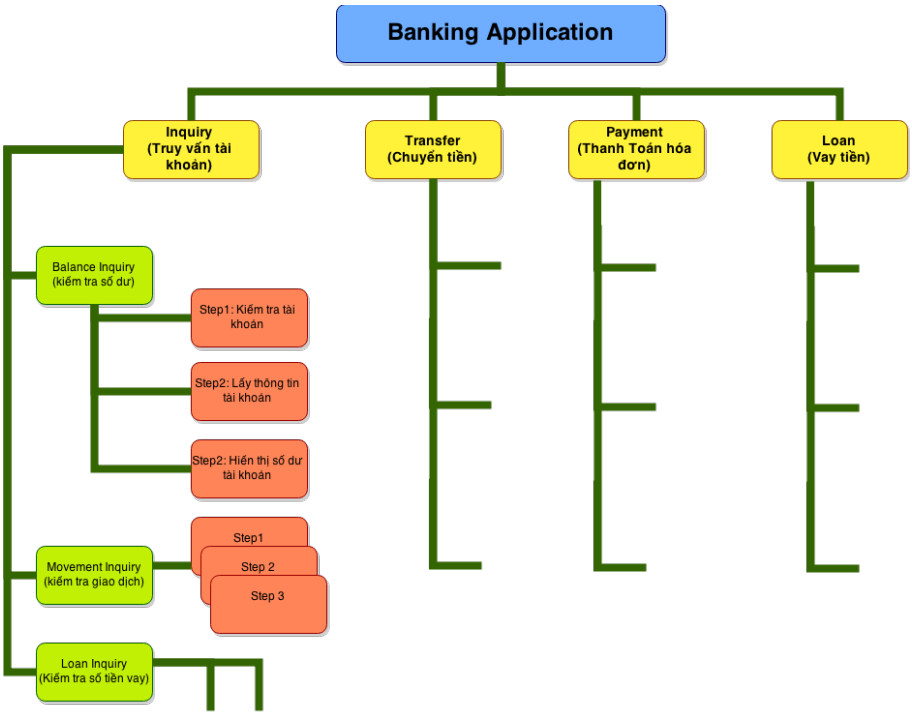


   Có rất nhiều khái niệm và lý thuyết về Structured Programming được phát biểu nhưng có 3 khái niệm chính mà mình muốn nói đến hôm nay, đó là top-down design, code reuse và information hiding.

   1. Top-Down design

      Top-down design là cách phân tích và phát triển ứng dụng dựa trên việc chia một bài toán lớn ra thành những bài toán nhỏ hơn, nhằm mục đích đơn giản hóa vấn đề. Khi ta giải quyết hết tất cả các bài toán nhỏ ấy, thì cả bài toàn tổng thể đều được giải quyết.

      Ví dụ cho dễ hiểu, bạn nhận được một dự án, yêu cầu xây dựng cả một hệ thống ngân hàng lớn. Bạn sẽ phân vân không biết sẽ bắt đầu từ đâu, viết như thế nào. Kỹ thuật Top-Down design có thể phân tích và chia nhỏ bài tóan ra như sau:



      Trong ví dụ trên mình chỉ phân tích chức năng trên BalanceInquiry mà thôi. Trên thực tế việc phân tích có thể kỹ càng hơn. Khi phân tích bài toán lớn ra thành những bài toán nhỏ hơn rồi, việc còn lại của bạn là hiện thực những phần nhỏ ấy, chúng ta sẽ được một ứng dụng lớn.

Việc chia một bài toán lớn thành những bài toán nhỏ sẽ giúp dễ dàng cho việc quản lý code, phân tích những chức năng lớn và những chức năng nhỏ hơn. Ngôn ngữ C cung cấp một cách thức để chia nhỏ ứng dụng ra thành nhiều phần, đó là function.

Nếu bạn để ý, bạn có thể thấy function main() đóng vài trò component cao nhất, các functon khác dc sử dụng trong main() giống như việc bạn đang chia nhỏ công việc cho các function khác (vd printf) thay vì phải tự hết tất cả trong hàm main.

2. Code reuse

   Code reuse là khả năng sử dụng lại đoạn code/chức năng đã được viết rồi trong một ứng dụng. Function trong C được thiết kế để có thể sử dụng lại mà không cần viết mới. Bạn có nhớ đã sử dụng nhiều lần printf trong khi viết không. Đấy chính là việc code reuse đấy.

3. Information hiding

   Information hiding là kỹ thuật che dấu đi sự phức tạp của một function đang phải thực hiện trong C. Khi bạn sử dụng function printf, bạn không cần quan tâm việc gì sảy ra trong function đó, bạn khôn biết rằng hàm printf kết nối với monitor như thế nào để print dữ liệu ra. Nhưng bạn vẫn có thể sử dụng hàm printf dễ dàng.

Chúng ta đã đi qua những tính chất của structured programming, và ta cũng thấy được tầm quan trọng function trong ngôn ngữ C. Phần tiếp theo mình sẽ hướng dẫn cách khai báo và sử dụng function trong C.

2. Function prototype

Trước khi xây nhà, ta thường sẽ coi trước bản thiết kế. C cũng cho phép ta xây dựng bản thiết kế của một function trước khi hiện thực nó, được gọi là function prototype. Cú pháp để tạo ra một function prototype như sau:

```
return_type   function_name( parameters );
```

- Return type: kiểu dữ liệu được trả về khi function này được thực hiện
- Function name: Tên của function
- Parameters: tham số truyền vào, là những giá trị mà function này cần để có thể thực thi

Ví dụ:

```
int congHaiSo(int soThu1, int soThu2);
```

Ví dụ trên, mình vừa định nghĩa ra prototype của một function có tên là congHaiSo (Cộng hai số), yêu cầu nhập vào 2 giá trị kiểu int, giá trị trả ra kiểu int. Mong muốn của mình là khi nhập 2 số đó vào, sẽ tự động tính toán ra tổng của 2 số đó.

Trong nhiều tình huống, ta không muốn định nghĩa kết quả trả về, có thể là do function đó không cần có kết quả, ta có thể sử dụng void. Void mang ý nghĩa là không có gì được trả về cả:

```
void printNumberToScreen(int number)
```

Hoặc một số function, ta không cần tham số tuyền vào, ta có thể đơn giản để trống:

```
void shutdownComputer()
```

Thông thường, function prototype được đặt ở dưới preprocessor:

```
#include <stdio.h>

int congHaiSo(int soThu1, int soThu2);
int nhanHaiSo(int soThu1, int soThu2);


int main() {
   printf("Nothing\n");
}
```

Trên thực tế, đối với một số ứng dụng lớn , function prototype được định nghĩa trong file header. Chúng ta sẽ nhắc tới vấn đề này ở phần cuối bài, mình sẽ để nó phía dưới preprocessor cho những ví dụ kế tiếp.

3. Function defination

Khi đã có bản thiết kế, chúng ta tiếp tục đến với việc thi công function. Đây là bước chúng ta hiện thực chức năng của function. Cú pháp của việc hiện thực này như sau:

```
return_type   function_name( parameters ) {
    // implement content
}
```

Bạn có thấy nó quen không nào, chính xác rồi, đây chính là cách bạn khai báo hàm main từ trước tới giờ đó. Một điều cần chú ý là function defination phải được đặt phía dưới của function prototype, không được để function defination trước prototype. Để hiểu rõ, ta tiếp tục với ví dụ sau đây:

```
#include <stdio.h>


int congHaiSo(int soThu1, int soThu2);
int nhanHaiSo(int soThu1, int soThu2);
int main() {
   printf("Nothing\n");
}
int congHaiSo(int soThu1, int soThu2) {
   int ketQua = suThu1 + soThu2;
   return ketQua;
}
int nhanHaiSo(int soThu1, int soThu2) {
   int ketQua = suThu1 * soThu2;
   return ketQua;
}
```

Ví dụ trên hiện thực function defination của congHaiSo và nhanHaiSo. Nội dung trong 2 function đó cũng tương tự như function main mà ta viết từ đầu tới giờ. Từ khóa return dùng để trả kết quả về cho người dùng (người gọi function này). Trong ví dụ trên, ta đã định nghĩa trước

kết quả trả về là kiểu int nên ta phải return một kết quả kiểu int.

Cũng như xây nhà, bạn không bắt buộc phải có bản thiết thì mới xây được nhà; C cho phép bạn tạo ra function defination ngay cả khi không có prototype. Ví dụ:

```
#include <stdio.h>
int main() {
   printf("Nothing\n");
}
int congHaiSo(int soThu1, int soThu2) {
   int ketQua = suThu1 + soThu2;
   return ketQua;
}
int nhanHaiSo(int soThu1, int soThu2) {
   int ketQua = suThu1 * soThu2;
   return ketQua;
}
```

Nhưng nếu bạn làm vậy, đến khi sử dụng function đó, bạn phải đảm bảo là function được defination trước khi sử dụng. Thực ra tôi khuyên bạn nên sử dụng prototype, thứ nhất là do prototype luôn nằm trên cùng, nên bạn không cần phải quan tâm việc đặt function defination trước khi sử dụng. Thứ hai là làm việc có bản thiết kế thì vẫn tốt hơn.

4. Function call

Sau khi đã có bản vẽ, đã hiện thực function rồi, thì đến lúc sử dụng function rồi. Thực ra, bạn đã sử dụng một function rất nhiều lần rồi mà bạn không biết, đó là function printf. Chúng ta đến tới ví dụ tiếp theo để hiểu rõ hơn nhé:

```
#include <stdio.h>

int congHaiSo(int soThu1, int soThu2);
int nhanHaiSo(int soThu1, int soThu2);

int main() {
   int number1 = 10;
   int number2 = 30;

   int resultOfCongHaiSo = congHaiSo(number1, number2);
   printf("Ket qua cua CongHaiSo: %d\n", resultOfCongHaiSo);

   printf("Ket qua cua CongHaiSo: %d\n", nhanHaiSo(number1, number2));
}

int congHaiSo(int soThu1, int soThu2) {
   int ketQua = soThu1 + soThu2;
   return ketQua;
}

int nhanHaiSo(int soThu1, int soThu2) {
   int ketQua = soThu1 * soThu2;
   return ketQua;
}
```

Và ta có kết quả là:

Ket qua cua CongHaiSo: 40
Ket qua cua CongHaiSo: 300

5. Variable scope

Varible scope, hay còn được gọi là tầm vực hoạt động của variable, định nghĩa khoảng thời thời gian có thể truy xuất được variable. C cung nhiều loại scope khác nhau tùy vào mục đích sử dụng.

1. Global variable

Còn được gọi là biến toàn cục, được định nghĩa bên ngoài function, và ta có thể truy xuất ở bất cứ đâu và bất cứ khi nào. Hiểu một cách khác là global variable tồn tại đến khi ứng dụng kết thúc.

Thường global variable được định nghĩa phía trên cùng của file:

```
#include <stdio.h>

double usdToVND(double usdValue);
double usdToVnRate = 21000;

int main() {

   double usd =  5;

   printf("Current USD to VND rate: %.1f\n", usdToVnRate);

   double vnd = usdToVND(usd);

   printf("%.1f USD is %.1f VND\n", usd, vnd);

}

double usdToVND(double usdValue) {

   int usdResult = usdValue * usdToVnRate;

   return usdResult;

}
```

Trong ví dụ trên, variable ***usdToVnRate*** được định nghĩa ngoài function main và usdToVND. Nó được xem như global variable và có thể được xài bất cứ đâu ở đâu. Chu kì sống của nó từ lúc nó được định nghĩa cho đến lúc ứng dụng hoàn tất và thoát.

Kết quả ta có:

Current USD to VND rate: 21000.0
5.0 USD is 105000.0 VND

2. Local variable

Ngược lại với global variable, local variable - biết cục bộ - được định nghĩa bên trong một function và chỉ có thể truy xuất khi còn đang trong function đó mà thôi. Khi thoát ra khỏi function, variable ấy không thể nào truy xuất được. Ta đến với ví dụ như sau.

```
#include <stdio.h>

void printLocalVariable();

int main() {

   int numberone = 1;
   printf("Function main, before invoke print, variable = %d\n", numberone);
   printLocalVariable();
   printf("Function main, before invoke print, variable = %d\n", numberone);

}

void printLocalVariable() {

   int numberone = 2;

   printf("Function printLocalVariable, variable = %d\n", numberone);

}
```

Kết quả ta có:

Function main, before invoke print, variable = 1
Function printLocalVariable, variable = 2
Function main, before invoke print, variable = 1

Như ta đã thấy, local variable ***numberone*** của function ***main*** và ***printLocalVariable*** tồn tại độc lập và không thể truy xuất của nhau được.

Ngoài ra, ta có thể định nghĩa local và global variable trùng tên với nhau:

```
#include <stdio.h>

void printLocalVariable();
int numberone = 5;

int main() {
   printf("Function main, before invoke print, variable = %d\n", numberone);
   printLocalVariable();
   printf("Function main, before invoke print, variable = %d\n", numberone);
}

void printLocalVariable() {
   int numberone = 2;
   printf("Function printLocalVariable, variable = %d\n", numberone);
}
```

Kết quả ta có:

Function main, before invoke print, variable = 5
Function printLocalVariable, variable = 2
Function main, before invoke print, variable = 5

3. Auto

   Là keyword được dùng cho local variable, nhằm mục đích xác định thời gian tồn tại của local variable chỉ tồn tại ở trong function. Thật ra keyword này rất ít khi được sử dụng vì lý do bất cứ local variable nào, đều được mặc định là được định nghĩa với keyword auto.

   Cách sử dụng keyword auto như nhau:

```
auto int localvariable;
```

4. Static for local variable

   Keyword static khi sử dụng với local variable, nó cho phép một local variable có thể giữ lại giá trị để có thể truy xuất lại nếu như function đó được sử dụng một lần nữa. Để hiểu rõ hơn về static, ta đến với ví dụ sau:

```
#include <stdio.h>

void printLocalVariable();
void printStaticLocalVariable();

int main() {
   printLocalVariable();
   printLocalVariable();
   printStaticLocalVariable();
   printStaticLocalVariable();
}

void printLocalVariable() {
   int numberone = 2;
   printf("Function printLocalVariable, variable = %d\n", numberone);
}

void printStaticLocalVariable() {
   static int numberone = 2;
   printf("Function printStaticLocalVariable, variable = %d\n", numberone);
   numberone = numberone + 5;
}
```

Và ta có kết quả:

```
Function printLocalVariable, variable = 2
Function printLocalVariable, variable = 2
Function printStaticLocalVariable, variable = 2
Function printStaticLocalVariable, variable = 7
Function printLocalVariable, variable = 2
Function printStaticLocalVariable, variable = 12
```

Static giúp cho local variable của function ***printStaticLocalVariable*** được giữ lại và tăng lên mỗi lần sử dụng.

Keyword static còn được sử dụng cho global variable và function, nhưng chúng ta sẽ nhắc tới vào cuối bài nhé.

5. Extern

Để viết hoàn chỉnh một ứng dụng lớn, thông thường người ta phải sử dụng nhiều file chứ không chỉ có một file như chúng ta hay làm từ đầu tới giờ. Vì vậy, việc chia sẽ variable và function giữa các file cần thiết. Keyword extern cho phép ta chia sẻ variable và function giữa các file. Thật ra tất cả các function trong C, mặc định đều được hiểu là có keyword extern nên function trong C mặc định là có thể chia sẻ giữa các file với nhau trong một ứng dụng.

Để hiểu rõ hơn, ta đến ví dụ sau, ta sẽ có 2 files, file 1 tên extern_main.c

```
#include<stdio.h>

extern int number1 = 10;
extern int number2 = 3;
extern int multiple2Values();

void main() {
   printf("Begin call function main() in extern_main.c\n");

   int result = multiple2Values();
   printf("** Result is %d\n", result);
}
```

File 2 tên là extern_math.c

```
#include<stdio.h>

extern int number1;
extern int number2;

extern int multiple2Values() {
   printf("Begin call function multiple2Values() in extern_math.c\n");
   return number1 * number2;
}
```

Để build 2 file vào một ứng dụng, ta dùng cú pháp sau của gcc:

```
gcc extern_main.c extern_math.c -o test
```

Và ta nhận được kết quả khi chạy:

Begin call function main() in extern_main.c
Begin call function multiple2Values() in extern_math.c
*** Result is 30*

Như ví dụ trên, function ***multiple2Values*** và variables ***number1, number2*** được chia sẽ giữa 2 file và sử dụng.

6. Static cho global variable và function

7. Register

   Thông thường, variable được lưu trữ và truy xuất thông qua bộ nhớ RAM, nhưng C cung cấp keyword register cho phép variable có thể được lưu trữ và truy xuất trên bộ nhớ register

của CPU (trong trường hợp hệ thống cho phép). Cú pháp để sử dụng register cũng đơn giản:

```
// register type name
register int numberOfDaysInWeek
```

Việc lưu trữ ở vùng nhớ register cho phép variable có thể truy xuất nhanh hơn so với cách thông thường.

8. Arrays

1. Arrays

Array là tập hợp nhiều variable có chung type (int, char, double, ...), được sắp xếp liên tục với nhau trên bộ nhớ (stack). array cho phép lập trình viên có thể lưu trữ và thao tác trên tập variable một cách dễ dàng và hiệu quả hơn. Để có thể thấy được lợi điểm của array, ta có thể tưởng tượng thay vì chúng ta khai bao 1000 variable để lưu trữ 1000 giá trị, ta có thể khai báo một array có 1000 phần tử, mọi việc sẽ đơn giản hơn nhiều.

Trước khi đến với cách sử dụng array, ta nên biết những tính chất của array bao gồm những điểm như sau:

- Những variables trong array được gọi là elements.
- Các elements trong array phải có cùng data type (kiểu dữ liệu).
- Các elements này được sử dụng và truy xuất bằng cùng một tên (tên này được định nghĩa cho cả array).
- Để thao tác trên một element trong array, ngôn ngữ C cung cấp index (chúng ta sẽ nhắc về khái niệm này khi vào những ví dụ).

Trước tiên, để khai báo một array, ta viết đoạn code sau:

```
int dictionary[5];
```

Đoạn code trên khai báo một array có tên là dictionary, với kiểu dữ kiệu int của các elements là int, và có thể chứa được 5 phần tử.

Tiếp theo, để khởi tạo giá trị cho một array. chúng ta có 2 cách.

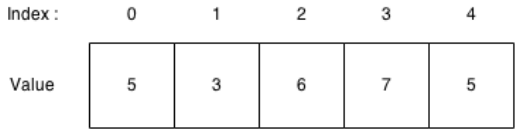
Cách 1: dùng để gán giá trị cho array lúc được tạo. Để vừa khai báo, vừa khởi tạo giá trị cho array, ta tham khảo đoạn code sau:

```
int dictionary[5] = {5,  3, 6, 7, 5};
```

Cách 2: dùng đẻ gán giá trị cho một array đã được khai báo rồi, ta dùng cú pháp với index:

```
dictionary[1] = 3;
```

Trong đoạn code trên, số 1 chính là index đùng để truy xuất element tại vị trí thứ 2 của Array. Bạn sẽ cảm thấy hơi rối vì sao index = 1 lại lấy element thứ 2 đúng không ? Đó là vì hệ thống index trong C (cũng như những ngôn ngữ lập trình khác) đều bắt đầu index bằng 0. Để hiểu rõ hơn bạn có thể xem hình sau:



Như hình ta thấy, index là 0 sẽ lấy giá trị đầu tiên, index là 1 sẽ lấy giá trị thứ 2 và cứ thế tiếp tục. Việc truy xuất dữ liệu cũng có thể dùng index như vậy:

```
printf("%d", dictionary[1]);
```

Một trường hợp khác mà chúng ta hay thao tác trên array đó là duyệt qua hết các elements trong array. Để làm việc đó, chúng ta thường dùng vòng lặp:

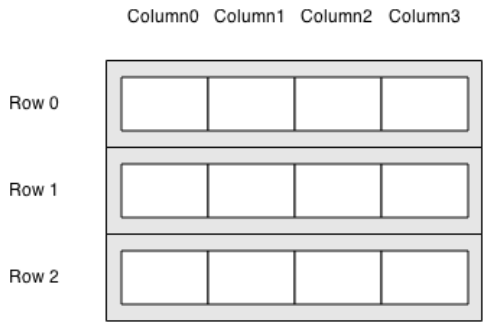
```
#include<stdio.h>

void printStudentId();
int studentIds[5]  = {10, 3, 7, 8, 9};

void main() {
   printStudentId();
   studentIds[3] = 2;
   printStudentId();
   return;
}

void printStudentId() {
   int index;
   for(index = 0; index < 5; index++) {
      printf("%d ", studentIds[index]);
   }
   printf("\n");
}
```

Kết quả ta có:

10 3 7 8 9
10 3 7 2 9

2. Multi-dimensional Arrays

Two Dimensional array, hay còn được gọi là mảng hai chiều, là một array mà mỗi elements của nó cũng là array nữa (có thể hiểu là array lồng vào array)



Để dễ hiểu hơn, bạn có thể tưởng tượng Two Dimensional arrays như một table với có hàng và cột. Để khởi tạo two dimensional arrays, ta sử dụng cú pháp sau đây:

```
int dictionary[3][4];
```

Dictionary ở trên được khai báo gồm 12 elements, theo dạng table thì sẽ có 3 cột và 4 dòng. Để khởi tạo giá trị cho two dimensional arrays, ta cũng làm gần giống như array thông thường:

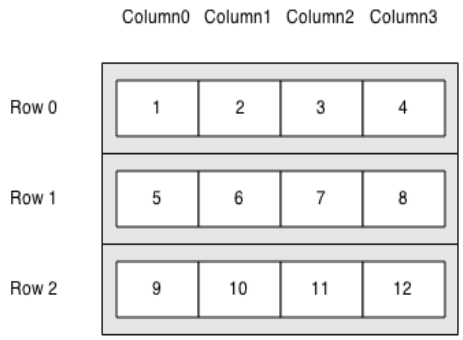
```
int dictionary[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8} , {9, 10, 11, 12} };
```

Sự khác biệt so với array thường là mỗi element của two dimensional array là một array nhỏ khác. Để gán giá trị cho two dimensional arrays, chúng ta phải dùng 2 index để xác định vị trí của element:

```
dictionary[1][0] = 5;
```

Trong ví dụ trên, chúng ta gán giá trị 5 vào vị trí cột 2, dòng 1. Cách lấy giá trị ra cũng giống tương đương như vậy:

```
printf("%d", dictionary[1][0]);
```



Để hiểu rõ hơn, ta đến với ví dụ sau:

C Programming language – Essential

C Programming language – Essential

C Programming language – Essential

C Programming language – Essential

```
#include<stdio.h>

void renderBoard();
char gameBoard[3][3] = {
                                { '-', '-', '-' },
                                { '-', '-', '-' },
                                { '-', '-', '-' }
                              };

void main() {
   gameBoard[0][0] = '1';
   renderBoard();
   gameBoard[1][0] = '0';
   renderBoard();
   gameBoard[1][1] = '1';
   renderBoard();
   gameBoard[2][0] = '0';
   renderBoard();
   gameBoard[2][2] = '1';
   renderBoard();
   return;
}

void renderBoard() {
   int indexRow, indexColumn;
   for(indexRow = 0; indexRow < 3; indexRow++) {
      for(indexColumn = 0; indexColumn < 3; indexColumn++) {
         printf("%c ", gameBoard[indexRow][indexColumn]);
      }
      printf("\n");
   }
   printf("\n");
}
```

Kết quả ta có:

1 - -

- - -

- - -

1 - -

0 - -

- - -

1 - -

0 1 -

- - -

1 - -

0 1 -

0 - -

1 - -

0 1 -

0 - 1

9. Pointers

1. Pointers
2. Address
3. Functions arguments
4. Address Arithmetic
5. Pointer Arrays; Pointers to Pointers
6. Pointers vs. Multi-dimensional Arrays
7. Pointers to Functions

10. Strings

1. Introduction
2. String Arrays
3. Manipulating Strings : strlen(), tolower(), toupper() , strcpy() , strcat()
4. Analyzing Strings : strcmp() , strstr()

11.Structures

1. Structures
2. Structures and Functions
3. Array of structures
4. Pointer to structures
5. Self-referential Structures
6. Table Lookup
7. Typedef
8. Unions
9. Bit fields

12. Dynamic Memory Allocation

1. Memory Concepts
2. Stack and Heap
3. sizeof
4. malloc()
5. calloc() and realloc()

13. Input & Output

1. Standard Input and Output
2. Formatted Output – printf
3. Formatted Input – Scanf
4. File Access
5. Error Handling - Stderr and Exit
6. Line Input and Output

14. The C Preprocessor

1. Header File
2. Static global variable and function
3. Function Definition File
4. main() Function File