

Tài liệu training lập trình C

06/2024 Tổng hợp các kiến thức cơ bản cần thiết

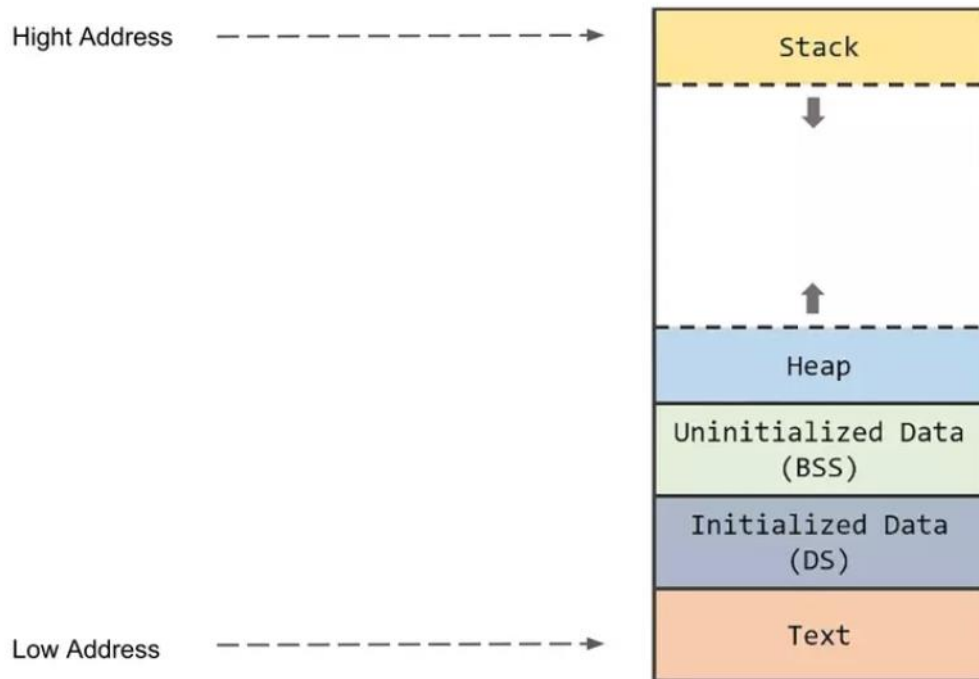


MỤC LỤC

1. Memory layout của chương trình C trên Ram.
2. Các bước để biên dịch 1 chương trình C thành file thực thi.
3. Các kiểu dữ liệu trong C, từ khóa static, volatile, typedef, struct, enum.
4. Cấp phát động, một số lỗi sử dụng bộ nhớ trong C
5. Xử lý hàm trong C, phân biệt tham chiếu và tham trị.
6. Cách tạo và thêm thư viện trong C.
7. Một số thuật toán xử lý trong C (mở rộng)

1. Memory layout của chương trình C trên Ram

Memory Layout



- **Stack**: Vùng nhớ dành cho các Function frame và biến cục bộ (local var)
- **Heap**: Vùng nhớ dành cho cấp phát động
- **BSS**: Vùng nhớ cho biến toàn cục (global var) và những biến tĩnh (static) không được khởi tạo tại giá trị ban đầu hoặc giá trị ban đầu bằng 0.
- **DS**: tương tự BSS nhưng được khởi tạo có giá trị ban đầu khác 0.
- **Text**: Vùng nhớ cho mã lệnh chương trình, là vùng nhớ ReadOnly

Link tham khảo: [Memory layout của một chương trình C/C++ - Viblo](#)

2. Các bước để biên dịch 1 chương trình C thành file thực thi.



Preprocessing

1. Tiền xử lý

Giai đoạn này bao gồm việc xử lý các chỉ thị tiền xử lý (`#include`, `#define`, `#if`, ...).

Preprocessor sẽ mở rộng các macro, xử lý các chỉ thị điều kiện, bao gồm nội dung của các file header và xóa các nội dung bình luận

- Kết quả: tạo ra file .i

Assembly

2. Hợp ngữ

Giai đoạn biên dịch chuyển đổi mã nguồn thành hợp ngữ (assembly code). Quá trình này tạo ra các file hợp ngữ (assembly files).

- Kết quả: tạo ra file .s

Compilation

3. Hợp ngữ

Giai đoạn hợp ngữ chuyển đổi mã hợp ngữ (assembly code) thành mã máy (machine code), tạo ra các file đối tượng (object files).

- Kết quả: tạo ra file .o

Linking

4. Liên kết

Giai đoạn liên kết kết hợp các file đối tượng và các thư viện liên kết với nhau để tạo ra một file thực thi.

- Kết quả: tạo ra file thực thi (executable)

3. Một số kiểu dữ liệu trong C

Kiểu số nguyên (Integer Types)

- char: Lưu trữ một ký tự đơn, thường là 1 byte. Cũng có thể dùng để lưu trữ số nguyên nhỏ.
- int: Lưu trữ số nguyên, thường là 4 byte.
- short: Lưu trữ số nguyên ngắn, thường là 2 byte.
- long: Lưu trữ số nguyên dài, thường là 4 hoặc 8 byte tùy hệ thống.
- long long: Lưu trữ số nguyên rất dài, thường là 8 byte.

Kiểu số thực (Floating Point Types)

- float: Lưu trữ số thực có độ chính xác đơn, thường là 4 byte.
- double: Lưu trữ số thực có độ chính xác kép, thường là 8 byte.
- long double: Lưu trữ số thực có độ chính xác kép mở rộng, thường là 8 hoặc 16 byte tùy hệ thống.

3. Một số kiểu dữ liệu trong C

Trong lập trình nhúng, ta có thể chạy chương trình trên nhiều nền tảng phần cứng khác nhau

→ Cần thống nhất kích thước của kiểu dữ liệu

Sử dụng thư viện chuẩn của C: `#include <stdint.h>`

`uint8_t, int8_t`: Kiểu dữ liệu số nguyên không và có dấu 8 bits

`uint16_t, int16_t`: Kiểu dữ liệu số nguyên không và có dấu 16 bits

`uint32_t, int32_t`: Kiểu dữ liệu số nguyên không và có dấu 32 bits

3. Một số kiểu dữ liệu trong C

Static: biến có từ khóa này sẽ giữ nguyên giá trị sau mỗi lần gọi hàm. Khi hàm hay biến có từ khóa này sẽ được giới hạn truy cập nội bộ, tức file khác sẽ không thể truy cập biến hay hàm có từ khóa này

Volatile: được sử dụng để báo cho trình biên dịch rằng giá trị của biến có thể thay đổi bất ngờ bởi các yếu tố bên ngoài chương trình (như phần cứng hoặc một luồng khác), và không được tối ưu hóa việc truy cập biến này.

```
#include <stdio.h>

int main()
{
    uint8_t a = 0;

    if(a != 0) {
        printf("Hello World");
    }

    return 0;
}
```

3. Một số kiểu dữ liệu trong C

Typedef: được sử dụng để định nghĩa lại tên kiểu dữ liệu, giúp mã nguồn dễ đọc hơn và thuận tiện hơn khi làm việc với các kiểu dữ liệu phức tạp.

Struct: (cấu trúc) được sử dụng để nhóm các biến có kiểu dữ liệu khác nhau lại thành một kiểu dữ liệu duy nhất.

```
typedef unsigned long ulong;  
typedef int* intptr;
```

```
ulong a = 1000000;
```

```
intptr p;
```

```
int b = 5;
```

```
p = &b;
```

```
printf("Value of a: %lu\n", a);
```

```
printf("Value of b: %d\n", *p);
```

```
struct Point {  
    int x;  
    int y;  
};
```

```
struct Point p1;
```

```
p1.x = 10;
```

```
p1.y = 20;
```

```
printf("Point p1: (%d, %d)\n", p1.x, p1.y);
```


3. Một số kiểu dữ liệu trong C

`typedef` và `struct` kết hợp

Bạn có thể kết hợp `typedef` với `struct` để định nghĩa lại tên kiểu dữ liệu cho cấu trúc.

Ví dụ:

```
c Copy code

typedef struct {
    int x;
    int y;
} Point;

Point p1;
p1.x = 10;
p1.y = 20;

printf("Point p1: (%d, %d)\n", p1.x, p1.y);
```




3. Một số kiểu dữ liệu trong C

`enum` trong C

`enum` (kiểu liệt kê) được sử dụng để định nghĩa một tập hợp các hằng số nguyên với tên gợi nhớ.

Ví dụ:

c

 Copy code

```
enum Color { RED, GREEN, BLUE };

enum Color myColor;
myColor = RED;

if (myColor == RED) {
    printf("The color is red.\n");
}
```



4. Cấp phát động

Cấp phát động (dynamic memory allocation) trong C là quá trình cấp phát bộ nhớ trong thời gian chạy (runtime) bằng cách sử dụng các hàm trong thư viện chuẩn C. Cấp phát động cho phép bạn quản lý bộ nhớ một cách linh hoạt, vì bạn có thể cấp phát và giải phóng bộ nhớ khi cần thiết.

Các hàm chính sử dụng cho cấp phát động trong C bao gồm malloc, calloc, realloc, và free, tất cả đều được khai báo trong thư viện <stdlib.h>.

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

int main() {
    int *ptr;
    ptr = (int *)calloc(5, sizeof(int)); // Cấp phát bộ nhớ cho mảng 5 số nguyên và khởi t

    if (ptr == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < 5; i++) {
        printf("%d ", ptr[i]); // In giá trị của mảng (sẽ là 0)
    }
    printf("\n");

    free(ptr); // Giải phóng bộ nhớ

    return 0;
}
```

4. Một số lỗi thường gặp

1. Memory Leak (Rò rỉ bộ nhớ)

Rò rỉ bộ nhớ xảy ra khi bộ nhớ được cấp phát động mà không được giải phóng. Điều này dẫn đến việc tiêu tốn bộ nhớ không cần thiết và có thể gây ra tình trạng thiếu bộ nhớ trong hệ thống.

Ví dụ:

```
c Copy code

#include <stdlib.h>

void memoryLeak() {
    int *ptr = (int *)malloc(sizeof(int) * 100);
    // Quên giải phóng bộ nhớ bằng free(ptr);
}

int main() {
    memoryLeak();
    return 0;
}
```

4. Một số lỗi thường gặp

```
c
#include <stdio.h>

int main() {
    int a[10]; // Khai báo mảng gồm 10 phần tử, với chỉ số từ 0 đến 9

    a[10] = 100; // Truy cập vượt ngoài giới hạn mảng

    printf("Value at a[10]: %d\n", a[10]);

    return 0;
}
```

5. Uninitialized Memory (Bộ nhớ chưa khởi tạo)

Sử dụng bộ nhớ chưa được khởi tạo có thể dẫn đến hành vi không xác định vì giá trị trong bộ nhớ có thể là ngẫu nhiên.

Ví dụ:

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

int main() {
    int *ptr = (int *)malloc(sizeof(int) * 100);
    printf("%d\n", ptr[0]); // In ra giá trị chưa được khởi tạo
    free(ptr);
    return 0;
}
```

4. Một số lỗi thường gặp

Đoạn chương trình này có lỗi không?

Nếu có thì là lỗi gì?

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

int main()
{
    char arr[] = "Hello World";
    char* a;
    strcpy(a, arr);
    printf("%s", a);

    return 0;
}
```

5. Hàm trong C

Cú pháp tổng quát của một hàm trong C:

c

Copy code

```
return_type function_name(parameter1_type parameter1, parameter2_type parameter2, ...) {  
    // Khai báo biến cục bộ (nếu cần thiết)  
  
    // Thực hiện các câu lệnh và logic trong hàm  
  
    // Trả về giá trị (nếu có)  
}
```

```
void printNumber(int num) {  
    printf("Number: %d\n", num);  
}
```

```
int getNumber() {  
    int num = 42;  
    return num;  
}
```

```
int add(int a, int b) {  
    int sum = a + b;  
    return sum;  
}
```

5. Hàm trong C

Tham trị: Đối số vào hàm sẽ được copy thành dữ liệu mới, sự thay đổi trong hàm sẽ không ảnh hưởng đến giá trị biến bên ngoài

1. Đặc điểm chính:

- Tham trị là cách truyền đối số vào hàm bằng cách sao chép giá trị của đối số vào tham số của hàm.
- Hàm sẽ làm việc với một bản sao của giá trị gốc, do đó các thay đổi trong hàm không ảnh hưởng đến giá trị gốc của biến được truyền vào.

2. Ví dụ:

```
c Copy code  
  
#include <stdio.h>  
  
// Hàm tính bình phương với tham trị  
void square(int num) {  
    num = num * num; // Thay đổi num trong hàm chỉ ảnh hưởng đến bản sao của num  
}
```


5. Hàm trong C

Tham chiếu: Đối số vào hàm là địa chỉ của biến, nên khi hàm thay đổi giá trị của đối số, giá trị của biến bên ngoài cũng sẽ bị thay đổi theo

- Tham chiếu là cách truyền đối số vào hàm bằng cách truyền địa chỉ của biến.
- Hàm sẽ làm việc trực tiếp với biến gốc thông qua địa chỉ này, do đó các thay đổi trong hàm sẽ ảnh hưởng trực tiếp đến giá trị gốc của biến được truyền vào.

2. Ví dụ (sử dụng con trỏ để tham chiếu):

```
c Copy code

#include <stdio.h>

// Hàm tính bình phương với tham chiếu
void square(int *num) {
    *num = (*num) * (*num); // Thay đổi giá trị của biến gốc thông qua con trỏ
}

int main() {
    int x = 5;
    square(&x); // Truyền địa chỉ của x vào hàm square
    printf("Modified value of x: %d\n", x); // Kết quả: Modified value of x: 25
    return 0;
}
```

Trong ví dụ này, hàm `square()` thay đổi giá trị của biến `x` trực tiếp thông qua tham chiếu (con trỏ).



5. Hàm trong C

Bình luận cho mỗi hàm, ta nên thống nhất sử dụng cấu trúc bình luận sau:

```
/**
 * @brief Initializes the USARTx peripheral according to the specified
 *        parameters in the USART_InitStruct .
 * @param USARTx: where x can be 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or
 *        UART peripheral.
 * @param USART_InitStruct: pointer to a USART_InitTypeDef structure that contains
 *        the configuration information for the specified USART peripheral.
 * @retval None
 */
```

```
/**
 * @brief Initializes the USARTx peripheral according to the specified
 *        parameters in the USART_InitStruct .
 * @param USARTx: where x can be 1, 2, 3, 4, 5, 6, 7 or 8 to select the USART or
 *        UART peripheral.
 * @param USART_InitStruct: pointer to a USART_InitTypeDef structure that contains
 *        the configuration information for the specified USART peripheral.
 * @retval None
 */
void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)
{
    uint32_t tmpreg = 0x00, apbclock = 0x00;
    uint32_t integerdivider = 0x00;
    uint32_t fractionaldivider = 0x00;
    RCC_ClocksTypeDef RCC_ClocksStatus;
```

5. Hàm trong C

Một số hàm thường sử dụng trong C với thư viện C tiêu chuẩn:

Xử lý chuỗi với thư viện string.h:

- `strlen(char* arr)`: Hàm trả về kích thước của chuỗi
- `strcpy(char* đích, char* nguồn)`: Copy dữ liệu chuỗi từ chuỗi nguồn đến chuỗi đích
- `strcmp(char* arr1, char* arr2)`: So sánh 2 chuỗi ký tự, nếu giống nhau trả về 0, trả về <0 nếu arr1 nhỏ hơn arr2 và ngược lại.
- `strtok(char* arr, const char* delim)`: Hàm tách chuỗi arr với các ký tự delim

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

int main() {
    // Chuỗi gốc cần tách
    char str[] = "Hello, world! Welcome to C programming.";


    // Chuỗi ký tự phân cách
    const char delim[] = " ,.!" ;

    // Con trỏ để nhận token
    char *token;

    // Lấy token đầu tiên
    token = strtok(str, delim);

    // Lấy các token tiếp theo
    while (token != NULL) {
        printf("%s\n", token);
        token = strtok(NULL, delim);
    }

    return 0;
}
```



5. Hàm trong C

Một số hàm thường sử dụng trong C với thư viện C tiêu chuẩn:

Xử lý dữ liệu

- `void *memcpy(void *dest, const void *src, size_t n)` được sử dụng để sao chép một vùng bộ nhớ từ địa chỉ nguồn đến địa chỉ đích.
- `int memcmp(const void *str1, const void *str2, size_t n)`; được sử dụng để so sánh hai vùng bộ nhớ.
- `void *memset(void *str, int c, size_t n)`; được sử dụng để thiết lập một vùng bộ nhớ với một giá trị cụ thể.

6. Tạo thư viện trong C

Thư viện viết bằng C bao gồm file mã nguồn .c và file header .h

File .c (Source File) chứa mã nguồn của các hàm và các định nghĩa biến mà bạn muốn biên dịch để chạy chương trình. Nó thường được sử dụng để:

- Định nghĩa các hàm: Bao gồm các hàm chức năng, các hàm phụ trợ, định nghĩa và khởi tạo biến.
- Viết logic chương trình chính: Trong trường hợp các chương trình nhỏ.

```
c Copy code  
  
// File: mylibrary.c  
  
#include "mylibrary.h"  
#include <stdio.h>  
  
int add(int a, int b) {  
    return a + b;  
}  
  
void greet() {  
    printf("Hello, World!\n");  
}
```

6. Tạo thư viện trong C

File .h chứa các khai báo hàm và các định nghĩa macro, cấu trúc dữ liệu, hằng số mà bạn muốn sử dụng trong các file .c khác. Nó thường được sử dụng để:

- Khai báo hàm: Định nghĩa các hàm và tham số của chúng mà bạn muốn sử dụng ở nhiều file .c.
- Khai báo biến toàn cục: Định nghĩa các biến toàn cục mà bạn muốn chia sẻ giữa các file.
- Khai báo macro, struct, enum: Định nghĩa các hằng số, cấu trúc dữ liệu, và enum.

c

Copy code

```
// File: mylibrary.h

#ifndef MYLIBRARY_H
#define MYLIBRARY_H

// Khai báo hàm
int add(int a, int b);
void greet();

#endif // MYLIBRARY_H
```

7. Một số thuật toán xử lý trong C

- Thuật toán sắp xếp: Selection sort, bubble sort,...
- Thuật toán lưu dữ liệu: Binary tree,...
- Thuật toán xử lý dữ liệu: Ring buffer, Linked List,...

Selection sort:

Ví dụ minh họa:

Giả sử có một mảng số nguyên đơn giản sau: `[64, 25, 12, 22, 11]`.

- Bước 1: Tìm phần tử nhỏ nhất từ vị trí đầu tiên đến hết mảng. Phần tử nhỏ nhất là 11 (tại vị trí cuối cùng).
- Bước 2: Hoán đổi phần tử nhỏ nhất (11) với phần tử ở vị trí đầu tiên của mảng (64).
- Bước 3: Di chuyển vị trí bắt đầu lên phía trước, giờ mảng là `[11, 25, 12, 22, 64]`.
- Bước 4: Tiếp tục tìm phần tử nhỏ nhất từ vị trí thứ hai đến hết mảng. Phần tử nhỏ nhất là 12 (tại vị trí thứ ba).
- Bước 5: Hoán đổi phần tử nhỏ nhất (12) với phần tử ở vị trí thứ hai của mảng (25).
- Bước 6: Di chuyển vị trí bắt đầu lên phía trước, giờ mảng là `[11, 12, 25, 22, 64]`.
- Và tiếp tục lặp lại quá trình này cho đến khi mảng được sắp xếp.

Kết quả cuối cùng của mảng sẽ là `[11, 12, 22, 25, 64]`.

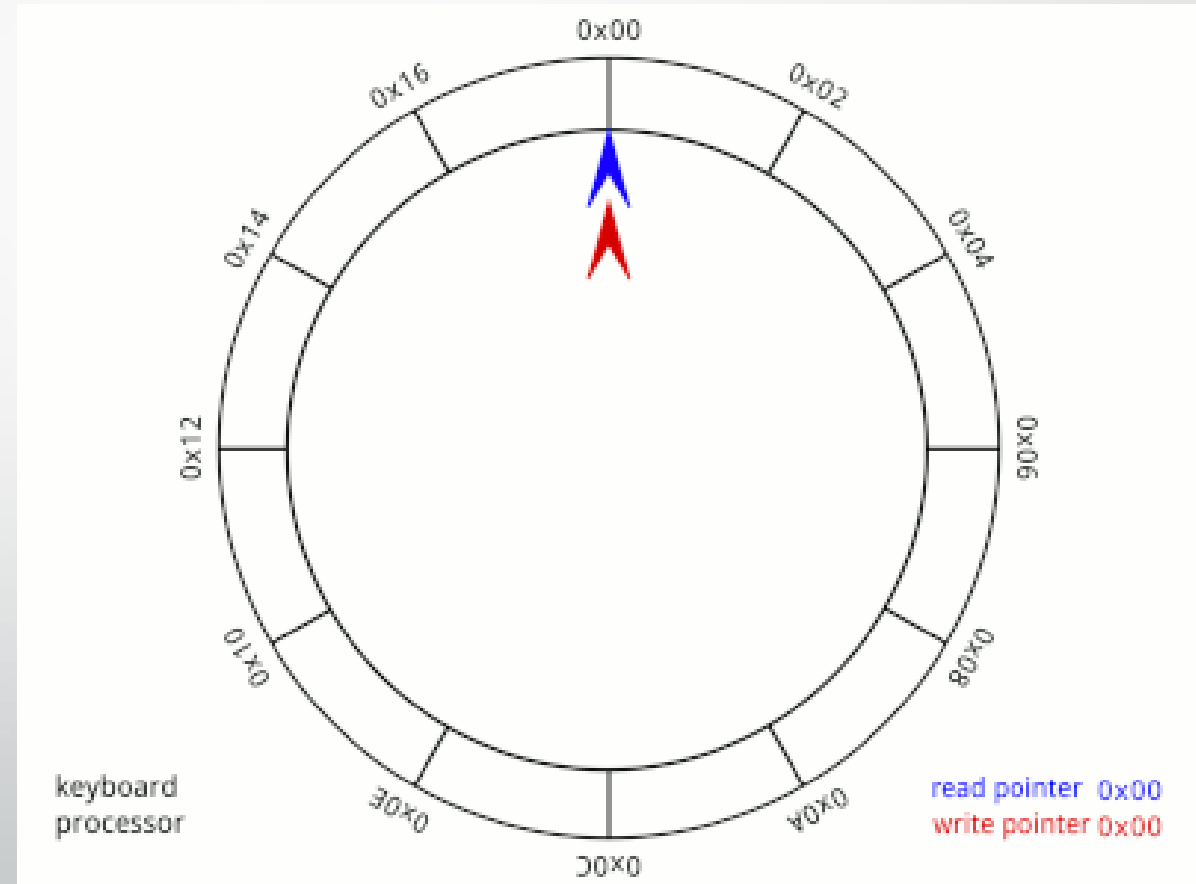
7. Một số thuật toán xử lý trong C

- Ring buffer:**
- Thuật toán sắp xếp: Selection sort, bubble sort,...
 - Thuật toán xử lý dữ liệu: Ring buffer, Linked List,...

```
// Cấu trúc của Ring Buffer
struct RingBuffer {
    int buffer[BUFFER_SIZE];
    int head;
    int tail;
};
```

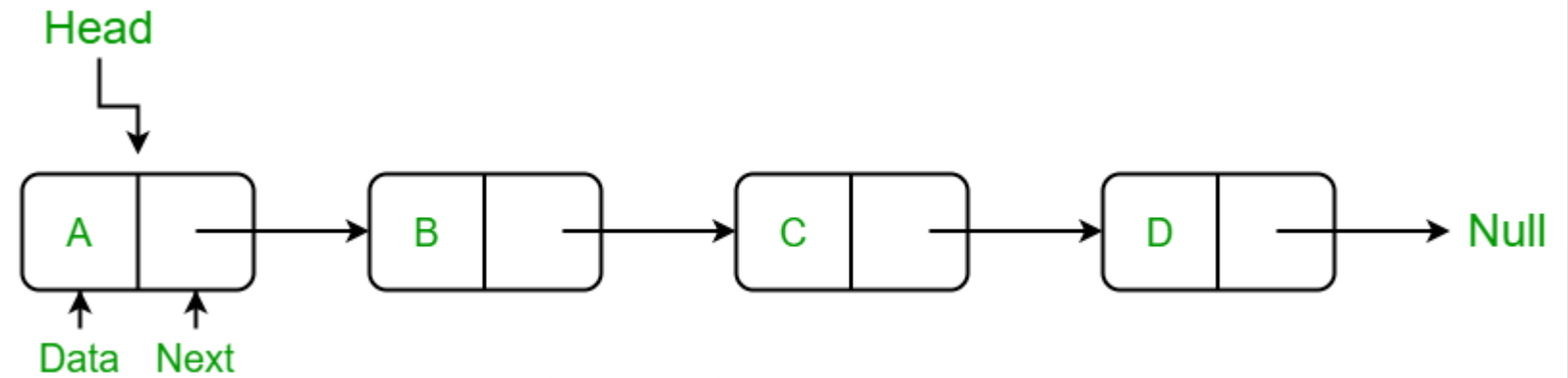
4. Các phép toán cơ bản:

- Push (Đẩy): Thêm một phần tử vào ring buffer.
- Pop (Lấy ra): Lấy ra phần tử từ ring buffer.
- Full (Đầy): Kiểm tra xem ring buffer có đầy hay không.
- Empty (Rỗng): Kiểm tra xem ring buffer có rỗng hay không.



7. Một số thuật toán xử lý trong C

Linked List:



c

```
struct Node {
    int data;
    struct Node *next;
};
```

c

```
struct Node *head = NULL; //
```

1. Thêm phần tử vào đầu danh sách (Insert at Beginning):

c

Copy code

```
void insertAtBeginning(int newData) {
    // Tạo một node mới
    struct Node *newNode = (struct Node *) malloc(sizeof(struct Node));
    newNode->data = newData;
    // Con trỏ tiếp theo của node mới trỏ đến head hiện tại
    newNode->next = head;
    // Cập nhật head để trỏ tới node mới
    head = newNode;
}
```