**Ngăn xếp Stack là gì? Cấu trúc và cơ chế hoạt động ra sao?**

Trong hướng dẫn này mình sẽ giới thiệu các bạn một cấu trúc lưu trữ được sử dụng nhiều trong lập trình đó chính là cấu trúc Stack. Đây là một cấu trúc lưu trữ với cơ chế khá là quen thuộc với chúng ta.



Chúng ta sẽ cùng nhau tìm hiểu cấu trúc Stack là gì? Và nó hoạt động như thế nào và có những thao tác nào đối với Stack.

**Mục lục**

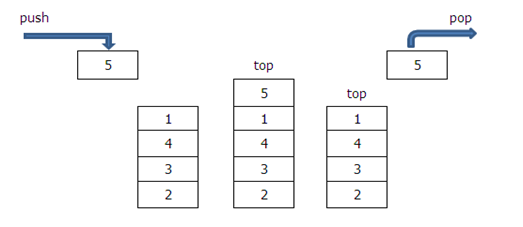
* [1. Ngăn xếp Stack là gì?](https://freetuts.net/ngan-xep-stack-la-gi-3146.html#goto-h2-0)
* [2. Mô tả Stack](https://freetuts.net/ngan-xep-stack-la-gi-3146.html#goto-h2-1)
  + [Mô tả Stack bằng mảng](https://freetuts.net/ngan-xep-stack-la-gi-3146.html#goto-h3-0)
  + [Mô tả Stack bằng danh sách liên kết đơn](https://freetuts.net/ngan-xep-stack-la-gi-3146.html#goto-h3-1)
* [3. Kết luận](https://freetuts.net/ngan-xep-stack-la-gi-3146.html#goto-h2-2)

**1. Ngăn xếp Stack là gì?**

Stack là một kiểu danh sách tuyến tính đặc biệt mà phép bổ sung và phép loại bỏ luôn luôn được thực hiện ở một đầu (gọi là đỉnh).

Hay còn có một định nghĩa khác: Ngăn xếp Stack là một cấu trúc dữ liệu trừu tượng làm việc theo nguyên lý vào sau ra trước LIFO (last in first out).

*Bài viết này được đăng tại [free tuts .net]*



**Ví dụ Stack trong thực tế**: Khi chúng ta bỏ các cái bánh vào một chiếc hộp, hành động bỏ vào tương tự như Push trong Stack, vì chúng ta sẽ bỏ lần lượt từng cái bánh vào trên đỉnh của chiếc hộp. Và khi ta muốn lấy bánh ra ta cũng sẽ phải lấy cái ở trên đỉnh ra trước, hành động này tương tự như Pop trong Stack.

Một ngăn xếp là một cấu trúc dữ liệu dạng thùng chứa (container) của các phần tử (thường được gọi là các Node). Có hai thao tác cơ bản là ***push***và ***pop***.

* **Push**bổ sung một phần tử vào đỉnh (top) của ngăn xếp, nghĩa là nó sẽ được thêm vào sau các phần tử đã có sẵn trong ngăn xếp.
* **Pop**giải phóng và trả về phần tử đang đứng ở đỉnh của ngăn xếp. Phần tử sau khi được lấy sẽ bị xóa khỏi ngăn xếp.

Ngoài ra, Stack cũng có một số thao tác bổ trợ khác:

* **isEmpty()**: Kiểm tra xem stack có rỗng hay không.
* **Top()**: Trả về giá trị của phần tử nằm ở đầu Stack mà không hủy nó khỏi Stack. Nếu Stack rỗng thì thông báo và không thực hiện được thao tác này.

**2. Mô tả Stack**

Trong phần này mình sẽ mô tả Stack bằng hai cách: mô tả Stack bằng mảng và mô tả Stack bằng danh sách liên kết đơn. Các bạn hãy xem hai cách này có gì giống và khác nhau nhé.

**Mô tả Stack bằng mảng**

Khi mô tả Stack bằng mảng, ta có một số đặc điểm sau:

* Việc bổ sung một phần tử vào Stack tương đương với việc thêm một phần tử ở cuối mảng.
* Việc loại bỏ một phần tử khỏi Stack tương đương với việc loại bỏ một phân tử ở cuối mảng.
* Stack sẽ bị tràn nếu bổ sung vào mảng đã đầy.
* Stack là rỗng khi số phần tử thực sự đang chưa trong mảng = 0.

**Mô tả Stack bằng danh sách liên kết đơn**

Khi mô tả Stack bằng danh sách liên kết đơn, ta cũng có một số đặc điểm sau:

* Việc bổ sung một phần tử vào Stack tương đương với việc thêm một phần tử vào cuối danh sách (insertlast).
* Việc loại bỏ một phần tử trong Stack cũng tương đương với việc loại bỏ một phần tử ở cuối danh sách.
* Stack bị tràn khi vùng không gian nhớ dùng cho các biến động không còn đủ để thêm một phần tử mới. Tuy nhiên việc kiểm tra này rất khó bởi nó phụ thuộc vào máy tính và ngôn ngữ lập trình. Vì vậy khi cài đặt ta có thể bỏ qua việc kiểm tra Stack tràn.

**3. Kết luận**

Như vậy là chúng ta đã tìm hiểu về cấu trúc Stack là gì? và cách hoạt động của nó theo cơ chế LIFO như thế nào. Đây là một cấu trúc được sử dụng rất nhiều trong lập trình vì vậy các bạn hãy tìm hiểu và học thật kỹ nó nhé. Ở bài tiếp theo mình sẽ hướng dẫn các bạn các thao tác trong Stack sử dụng mảng, hãy chú ý theo dõi !!!

**Cài đặt Stack bằng danh sách liên kết**

Trong hướng dẫn này mình sẽ giới thiệu các bạn cách cài đặt cấu trúc Stack trong danh sách liên kết. Đây là một trong hai cách được sử dụng để cài đặt cấu trúc Stack.



Chúng ta sẽ thực hiện lần lượt các thao tác trong Stack sử dụng danh sách liên kết để cài đặt: Push, Pop, Top(), isEmpty().

**Mục lục**

* [1. Cấu trúc dữ liệu của Stack](https://freetuts.net/cai-dat-stack-bang-danh-sach-lien-ket-3147.html#goto-h2-0)
* [2. Hàm isEmpty](https://freetuts.net/cai-dat-stack-bang-danh-sach-lien-ket-3147.html#goto-h2-1)
* [3. Hàm Push](https://freetuts.net/cai-dat-stack-bang-danh-sach-lien-ket-3147.html#goto-h2-2)
* [4. Hàm Pop](https://freetuts.net/cai-dat-stack-bang-danh-sach-lien-ket-3147.html#goto-h2-3)
* [5. Hàm Top](https://freetuts.net/cai-dat-stack-bang-danh-sach-lien-ket-3147.html#goto-h2-4)
* [6. Ví dụ xây dựng một cấu trúc Stack hoàn chỉnh](https://freetuts.net/cai-dat-stack-bang-danh-sach-lien-ket-3147.html#goto-h2-5)
* [7. Kết luận](https://freetuts.net/cai-dat-stack-bang-danh-sach-lien-ket-3147.html#goto-h2-6)

**1. Cấu trúc dữ liệu của Stack**

Trong phần này mình sử dụng danh sách liên kết để cài đặt cho Stack vì vậy cấu trúc dữ liệu chính của nó chính là danh sách liên kết. Ta sẽ có cấu trúc của một Node trong Stack.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | //khai báo cấu trúc Node  struct Node {  **int** data;      Node \*next;  };  typedef struct Node \*stack; |

Tiếp đến ta sẽ có cấu trúc của Stack, để quản lý Stack ta sử dụng con trỏ đầu để quản lý.

*Bài viết này được đăng tại [free tuts .net]*

|  |  |
| --- | --- |
| 1  2  3  4  5 | //khai báo cấu trúc stack  struct Stack{      Node \*head;      Node \*tail;  }; |

Sau khi đã khởi tạo cấu trúc của một Node và cấu trúc của Stack, ta sẽ tạo một Node mới.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | //tạo Node  Node \*createNode(**int** data) {      Node \*p = new Node();      if (p == NULL) {          return NULL;      }      p->data = data;      p->next = NULL;        return p;  } |

Như vậy là ta dã có cấu trúc cần thiết trong Stack cũng như đã khởi tạo cho nó, bây giờ chúng ta đã có thể thực hiện các thao tác trên Stack vừa được tạo.

**2. Hàm isEmpty**

Việc đầu tiên khi muốn thực hiện một thao tác nào đó trong Stack thì ta phải kiểm tra xem trong Stack có tồn tại phẩn tử hay không, Nếu Stack rỗng thì ta không thể thực hiện các thao tác đó trên Stack được.

|  |  |
| --- | --- |
| 1  2  3  4 | //kiểm tra stack rỗng  **bool** isEmpty(stack s) {      return (s == NULL);  } |

**3. Hàm Push**

Khi thêm một phần tử vào Stack, trước tiên ta cần kiểm tra xem danh sách có rỗng hay không, nếu danh sách rỗng thì phần tử đầu tiên thêm vào chính là Node p. Ngược lại nếu danh sách đã tồn tại phần tử thì ta cho con trỏ của Node p trỏ đến Node đầu của danh sách để tạo liên kết.

Sau khi tạo liên kết cho Node p vừa được thêm vào, ta cập nhật là Node đầu cho danh sách.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | void push(stack &s, int data) {    // tạo một Node mới      Node \*ptr = createNode(data);    // kiểm tra stack rỗng      if (isEmpty(s)) {          s = ptr;      }    //nếu tồn tại phần tử ta trỏ đến phần tử đầu danh sách      else {          ptr->next = s;          s = ptr;      }  } |

**4. Hàm Pop**

Việc xóa một phần tử khỏi Stack được thực hiện khá đơn giản, đầu tiên ta cũng sẽ phải kiểm tra xem danh sách có tồn tại phần tử hay không. Nếu danh sách rỗng thì ta thực hiện return và kết thúc hàm, ngược lại nếu danh sách có tồn tại phần tử thì ta bắt đầu thực hiện các bước sau:

* Gán giá trị của Node đầu Stack vào biến x.
* Cập nhập Node đầu Stack và Node tiếp theo.
* Xóa Node đầu Stack vừa lấy chính là Node x.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | int pop(stack &s) {      if (!isEmpty(s)) {      //tạo một biến data          int data = s->data;      //tạo Node x và gán cho bằng s          Node \*x = s;      // cho s = s trỏ đến next          s = s->next;      // thực hiện xóa node x          delete(x);      cout<<"Xóa thành công !!";          return data;      }      else {          cout << "Stack rỗng!" << endl;      }  } |

**5. Hàm Top**

Hàm Top() có chức năng hiển thị phần tử và không xóa nó khỏi Stack, không giống như Pop sau khi lấy phần tử từ Stack ra nó sẽ thực hiện xóa luôn phần tử đó khỏi Stack.

Đầu tiên ta sẽ kiểm tra xem danh sách có phần tử hay không, nếu có thì ta mới thực hiện lấy giá trị x, nếu danh sách rỗng thì return và kết thúc hàm.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | int top(stack s) {          // kiểm tra stack tồn tại phần tử thì thực hiện hiển thị ra màn hình      if (!isEmpty(s)) {          return s->data;      }      else {          cout << "Stack rỗng!" << endl;      }  } |

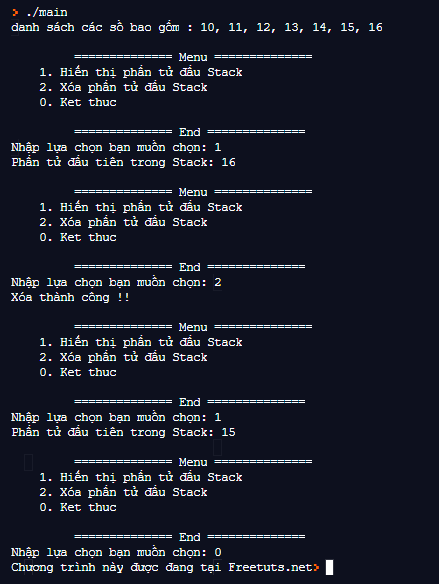
**6. Ví dụ xây dựng một cấu trúc Stack hoàn chỉnh**

Ta đã có lần lượt các hàm cần thiết cho một cấu trúc Stack, các bạn có thể tham khảo một chương trình mình đã viết sẵn dưới đây. Chương trình chỉ đơn giản là thêm phần tử, xóa phần tử và hiển thị phần tử và sử dụng danh sách liên kết để cài đặt cho Stack.

**Full code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105 | #include<iostream>  using namespace std;  //khai báo cấu trúc Node  struct Node {  **int** data;      Node \*next;  };  typedef struct Node \*stack;  //khai báo cấu trúc stack  struct Stack{      Node \*head;      Node \*tail;  };    //kiểm tra stack rỗng  **bool** isEmpty(stack s) {      return (s == NULL);  }    //tạo Node  Node \*createNode(**int** data) {      Node \*p = new Node();      if (p == NULL) {          return NULL;      }      p->data = data;      p->next = NULL;        return p;  }    void push(stack &s, **int** data) {    // tạo một Node mới      Node \*ptr = createNode(data);    // kiểm tra stack rỗng      if (isEmpty(s)) {          s = ptr;      }    //nếu tồn tại phần tử ta trỏ đến phần tử đầu danh sách      else {          ptr->next = s;          s = ptr;      }  }    **int** top(stack s) {      if (!isEmpty(s)) {          return s->data;      }      else {          cout << "Stack is empty!" << endl;      }  }      **int** pop(stack &s) {      if (!isEmpty(s)) {      //tạo một biến data  **int** data = s->data;      //tạo Node x và gán cho bằng s          Node \*x = s;      // cho s = s trỏ đến next          s = s->next;      // thực hiện xóa node x          delete(x);      cout<<"Xóa thành công !!";          return data;      }      else {          cout << "Stack rỗng!" << endl;      }  }    **int** main() {  **int** lc, k;      stack s;    push(s, 10);    push(s, 11);    push(s, 12);    push(s, 13);    push(s, 14);    push(s, 15);    push(s, 16);    cout<<"danh sách các số bao gồm : 10, 11, 12, 13, 14, 15, 16";  do{          cout << "\n\n\t\t ============== Menu ==============";          cout << "\n\t1. Hiển thị phần tử đầu Stack";          cout << "\n\t2. Xóa phần tử đầu Stack";          cout << "\n\t0. Ket thuc";          cout << "\n\n\t\t ============== End ==============";      cout<<"\nNhập lựa chọn bạn muốn chọn: ";          cin>> lc;          switch(lc){              case 0: break;              case 1:        cout<<"Phần tử đầu tiên trong Stack: "<<top(s);        break;              case 2:        pop(s);        break;              default: cout<<"\nNhập sai, vui lòng nhập lại!";          }      } while(lc);    cout<<"Chương trình này được đang tại Freetuts.net";  } |

**Kết quả:** Mình đã thực hiện thêm phần tử sẵn trước đó bao gồm các số: 10, 11, 12, 13 ,14, 15, 16.



**7. Kết luận**

Như vậy là chúng ta đã tìm hiểu xong về cấu trúc dữ liệu của Stack và các thao tác cơ bản của nó. Trong bài này mình đã sử dụng danh sách liên kết để cài đặt cho Stack, ở bài tiếp theo mình sẽ sử dụng mảng một chiều để cài đặt cho Stack. Các bạn hãy chú ý theo dõi nhé !!!

**Cài đặt Stack bằng mảng một chiều**

Trong hướng dẫn này mình sẽ giới thiệu các bạn cách để tạo Stack bằng mảng một chiều. Ở bài trước chúng ta đã tìm hiểu cách cài đặt Stack bằng danh sanh liên kết rồi. Đây là hai cách cơ bản nhất để có thể cài đặt Stack.



Chúng ta sẽ lần lượt thực hiện tạo các hàm cơ bản cho Stack như: **isEmpty()**, **isFull()**, **push()**,**top()**, **pop()**. Đây là các hàm cơ bản nhưng rất cần thiết cho Stack, hầu hết khi làm việc với Stack cần phải có các hàm này.

**Mục lục**

* [1. Hàm isEmpty và isFull](https://freetuts.net/cai-dat-stack-bang-mang-mot-chieu-3169.html#goto-h2-0)
  + [Hàm isEmpty](https://freetuts.net/cai-dat-stack-bang-mang-mot-chieu-3169.html#goto-h3-0)
  + [Hàm isFull](https://freetuts.net/cai-dat-stack-bang-mang-mot-chieu-3169.html#goto-h3-1)
* [2. Hàm Push](https://freetuts.net/cai-dat-stack-bang-mang-mot-chieu-3169.html#goto-h2-1)
* [3. Hàm Top](https://freetuts.net/cai-dat-stack-bang-mang-mot-chieu-3169.html#goto-h2-2)
* [4. Hàm Pop](https://freetuts.net/cai-dat-stack-bang-mang-mot-chieu-3169.html#goto-h2-3)
* [5. Ví dụ xây dựng một cấu trúc Stack hoàn chỉnh](https://freetuts.net/cai-dat-stack-bang-mang-mot-chieu-3169.html#goto-h2-4)
* [6. Kết luận](https://freetuts.net/cai-dat-stack-bang-mang-mot-chieu-3169.html#goto-h2-5)

**1. Hàm isEmpty và isFull**

Trước khi tạo các hàm isEmpty() và isFull() để kiểm tra Stack rỗng hay đầy thì ta phải tạo một số biến cơ bản cần sử dụng trong Stack:

* **curren\_size**: chúng ta sẽ khởi tạo cho nó bằng -1, đây là size hiện tại của Stack.
* **MAX\_SIZE**: chúng ta sẽ khởi tạo cho nó bằng 100, đây là size tối đa của Stack.
* **stack[MAX\_SIZE]**: đây là một mảng stack với số phần tử tối đa là MAX\_SIZE.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | /\* tạo các biến cơ bản \*/  //tạo size hiện tại và khởi tạo cho nó bằng -1  **int** curren\_size = -1;  //tạo một hằng số làm size max = 100  const **int** MAX\_SIZE = 100;  //tạo một mảng stack với số phần tử bằng max  **int** stack[MAX\_SIZE]; |

**Hàm isEmpty**

Hàm kiểm tra Stack có tồn tại phần tử hay không là một hàm rất quan trọng. Vì khi muốn thực hiện các thao tác như xóa đối với Stack ta cần biết được trong Stack có tồn tại phần tử hay không, nếu không tồn tại thì ta không thể xóa phần tử trong Stack được.

*Bài viết này được đăng tại [free tuts .net]*

Để kiểm tra Stack rỗng hay không rất đơn giản, vì lúc đầu ta đã khởi tạo cho biến curren\_size = -1, vậy nên ta chỉ cần kiểm tra nếu curren\_size = -1 thì tức là trong Stack chưa có gì, khi đó ta chỉ cần return và kết thúc hàm.

|  |  |
| --- | --- |
| 1  2  3  4  5 | /\* kiểm tra stack rỗng \*/  **bool** isEmpty(){    //kiểm tra nếu curren\_size == -1 thì return và kết thúc hàm    return (curren\_size == -1);  } |

**Hàm isFull**

Tương tự như hàm kiểm tra Stack rỗng hay không, việc kiểm tra Stack đã đầy hay chưa cũng rất quan trọng. Vì khi muốn thêm một phần tử nào đó vào Stack ta cần biết được trong Stack còn chỗ trống hay không, nếu Stack đã đầy thì ta không thể thêm phần tử vào Stack được.

Để kiểm tra Stack đã đầy hay chưa ta chỉ cần kiểm tra nếu curren\_size == MAX\_SIZE thì ra return rồi kết thúc hàm.

|  |  |
| --- | --- |
| 1  2  3  4  5 | /\* kiểm tra stack đầy \*/  **bool** isFull(){    //kiểm tra neeys curren\_size == MAX\_SIZE thì return và kết thúc hàm    return (curren\_size == MAX\_SIZE);  } |

**2. Hàm Push**

Sau khi đã tạo các hàm điều kiện isEmpty() và isFull() ta sẽ bắt đầu thực hiện tạo hàm thêm phần tử vào Stack. Nếu không có dữ liệu ta không thể thực hiện các thao tác khác trên Stack được, vì vậy đây là một bước cũng khá quan trọng.

Như đã nói thì để có thể thêm một phần tử vào Stack thì trong Stack vẫn còn chỗ trống, vì vậy trước tiên ta cần sử dụng hàm isFull() để kiểm tra xem Stack đã đầy hay chưa, nếu chưa đầy thì ta mới thực hiện các câu lệnh.

* Tăng curren\_size lên để tạo thêm chỗ trống trong stack để thêm phần tử mới vào.
* Gán data vào vị trí curren\_size trong Stack.

Nếu trong Stack đã đầy thì thông báo Stack đã đầy và kết thúc hàm.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | /\* hàm thêm phần tử vào Stack \*/  void push(**int** data){    //việc đầu tiên ta kiểm tra xem stack đã đầy hay chưa, nếu chưa tha thực hiện:    if(!isFull()){      //tăng curren\_size lên để tạo thêm chổ trống trong stack để thêm phần tử mới vào      curren\_size++;      //sau đó gán data vào đúng vị trí curren\_size trong stack      stack[curren\_size] = data;    }    //ngược lại nếu trong stack đã đầy thì thông báo cho người dùng biết rằng stack đã đầy    else{      cout<<"Stack đã đầy !!"<<endl;    }  } |

**3. Hàm Top**

Hàm top() là một hàm lấy giá trị ở trên đầu (top) trong Stack nhưng không xóa nó khỏi Stack, giống như ta chỉ xem phần tử đầu thôi chứ không xóa nó.

Việc đầu tiên ta sẽ kiểm tra xem Stack có tồn tại phần tử hay không, nếu Stack có tôn tại phần tử thì ta bắt đầu thực hiện các câu lệnh:

* Gán giá trị ở vị trí curren\_size cho biến data.
* Return giá trị data

Ngược lại nếu trong Stack rỗng thì thông báo và kết thúc hàm.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | /\* lấy phần tử ở top nhưng không xóa nó khỏi stack \*/  **int** top(){    //kiểm tra xem stack có rỗng hay không, nếu không ta thực hiện:    if(!isEmpty()){      //gán giá trị ở vị trí curren\_size cho biến data  **int** data = stack[curren\_size];      //sau đó return data      return data;    }    //ngược lại nếu stack rỗng thì thông báo cho người dùng biết stack rỗng    else{      cout<<"Stack rỗng !!"<<endl;    }  } |

**4. Hàm Pop**

Tương tự như hàm top(), hàm pop() cũng có nhiệm vụ là lấy phần tử trên cùng (top) trong Stack nhưng đồng thời sẽ xóa nó khỏi Stack. Đây là điểm khác nhau giữa hai hàm lấy phần tử này.

Ta cũng sẽ phải kiểm tra xem Stack có tồn tại phần tử hay không, nếu có tồn tại phần tử thì ta thực hiện các câu lệnh:

* Gán giá trị ở vị trí curren\_size cho biến data.
* Giảm curren\_size đi vì hàm pop() sau khi lấy sẽ xóa phần tử đó khỏi Stack.
* Return giá trị data

Ngược lại nếu Stack rỗng thì thông báo và kết thúc hàm.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | /\*lấy phần tử ở top và xóa nó khỏi stack\*/  **int** pop(){    //kiểm tra xem stack có rỗng hay không, nếu không ta thực hiện:    if(!isEmpty()){      //gán giá trị ở vị trí curren\_size cho biến data  **int** data = stack[curren\_size];      //giảm curren\_size đi vì hàm pop() sau khi lấy sẽ xóa phần tử đó khỏi stack      curren\_size--;      //sau đó return data      return data;    }    //ngược lại nếu stack rỗng thì thông báo cho người dùng biết stack rỗng    else{      cout<<"Stack rỗng !!"<<endl;    }  } |

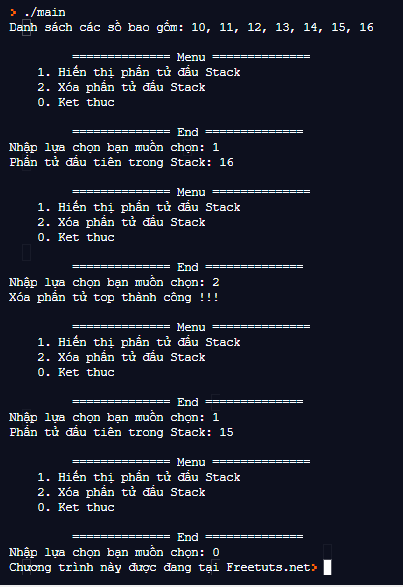
**5. Ví dụ xây dựng một cấu trúc Stack hoàn chỉnh**

Trong ví dụ này mình sẽ thực hiện tạo một mảng stack[] là các số nguyên, sau đó thêm một số phần tử sẵn trong Stack. Mình sẽ thực hiện các thao tác như hiển thị giá trị trên cùng, xóa giá trị trên cùng trong Stack. Các bạn có thể thao khảo code dưới đây, trong đó mình có chú thích cho các bạn dễ hiểu.

**Full code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102 | #include <iostream>  using namespace std;  /\* tạo các biến cơ bản \*/  //tạo size hiện tại và khởi tạo cho nó bằng -1  **int** curren\_size = -1;  //tạo một hằng số làm size max = 100  const **int** MAX\_SIZE = 100;  //tạo một mảng stack với số phần tử bằng max  **int** stack[MAX\_SIZE];    /\* kiểm tra stack rỗng \*/  **bool** isEmpty(){    //kiểm tra nếu curren\_size == -1 thì return và kết thúc hàm    return (curren\_size == -1);  }    /\* kiểm tra stack đầy \*/  **bool** isFull(){    //kiểm tra neeys curren\_size == MAX\_SIZE thì return và kết thúc hàm    return (curren\_size == MAX\_SIZE);  }    /\* hàm thêm phần tử vào Stack \*/  void push(**int** data){    //việc đầu tiên ta kiểm tra xem stack đã đầy hay chưa, nếu chưa tha thực hiện:    if(!isFull()){      //tăng curren\_size lên để tạo thêm chổ trống trong stack để thêm phần tử mới vào      curren\_size++;      //sau đó gán data vào đúng vị trí curren\_size trong stack      stack[curren\_size] = data;    }    //ngược lại nếu trong stack đã đầy thì thông báo cho người dùng biết rằng stack đã đầy    else{      cout<<"Stack đã đầy !!"<<endl;    }  }    /\* lấy phần tử ở top nhưng không xóa nó khỏi stack \*/  **int** top(){    //kiểm tra xem stack có rỗng hay không, nếu không ta thực hiện:    if(!isEmpty()){      //gán giá trị ở vị trí curren\_size cho biến data  **int** data = stack[curren\_size];      //sau đó return data      return data;    }    //ngược lại nếu stack rỗng thì thông báo cho người dùng biết stack rỗng    else{      cout<<"Stack rỗng !!"<<endl;    }  }    /\*lấy phần tử ở top và xóa nó khỏi stack\*/  **int** pop(){    //kiểm tra xem stack có rỗng hay không, nếu không ta thực hiện:    if(!isEmpty()){      //gán giá trị ở vị trí curren\_size cho biến data  **int** data = stack[curren\_size];      //giảm curren\_size đi vì hàm pop() sau khi lấy sẽ xóa phần tử đó khỏi stack      curren\_size--;      //sau đó return data      return data;    }    //ngược lại nếu stack rỗng thì thông báo cho người dùng biết stack rỗng    else{      cout<<"Stack rỗng !!"<<endl;    }  }        **int** main() {  **int** lc;    //ta thực hiện thêm một vài phần tử vào stack    //cụ thể ta sẽ thêm những số sau: 10, 11, 12, 13, 14, 15, 16    cout<<"Danh sách các số bao gồm: 10, 11, 12, 13, 14, 15, 16";    for(**int** i = 10; i <= 16; i++){      push(i);    }    do{          cout << "\n\n\t\t ============== Menu ==============";          cout << "\n\t1. Hiển thị phần tử đầu Stack";          cout << "\n\t2. Xóa phần tử đầu Stack";          cout << "\n\t0. Ket thuc";          cout << "\n\n\t\t ============== End ==============";          cout<<"\nNhập lựa chọn bạn muốn chọn: ";          cin>> lc;          switch(lc){              case 0: break;              case 1:                cout<<"Phần tử đầu tiên trong Stack: "<<top();                break;              case 2:                pop();                cout<<"Xóa phần tử top thành công !!!";                break;              default:                cout<<"\nNhập sai, vui lòng nhập lại!";          }      } while(lc);    cout<<"Chương trình này được đang tại Freetuts.net";  } |

**Kết quả:**Mình đã khởi tạo sẵn các giá trị 10, 11, 12, 13, 14, 15, 16 trong Stack.



**6. Kết luận**

Như vậy là chúng ta đã tìm hiểu xong cách cài đặt Stack bằng mảng một chiều, các bạn hãy nắm rõ cách cài đặt Stack bằng cả mảng và cả danh sách liên kết nhé. Vì mỗi bài toán ta cần sử dụng một cấu trúc lưu trữ khác nhau, vậy nên hãy sử dụng nó một cách linh hoạt. Ở các bài tiếp theo mình sẽ áp dụng Stack để giải một số bài toán cơ bản trong C++, các bạn hãy chú ý theo dõi nhé !!!

**Bài tập chuyển đổi cơ số bằng Stack**

Trong hướng dẫn này mình sẽ thực hiện giải một bài toán chuyển đổi cơ số áp dụng Stack. Đây là một bài toán rất phổ biến trong lập trình, để làm được bài này các bạn cần nắm rõ quy tắc chuyển đổi giữa các cơ số.



Chúng ta sẽ cùng nhau thực hiện một chương trình đổi cơ số thập phân sang cơ số nhị phân (2), bát phân (8) và thập lục phân (16).

**Mục lục**

* [1. Gợi ý cách thực hiện](https://freetuts.net/chuyen-doi-co-so-bang-stack-3170.html#goto-h2-0)
* [2. Ứng dụng Stack để chuyển đổi cơ số](https://freetuts.net/chuyen-doi-co-so-bang-stack-3170.html#goto-h2-1)
* [3. Kết luận](https://freetuts.net/chuyen-doi-co-so-bang-stack-3170.html#goto-h2-2)

**1. Gợi ý cách thực hiện**

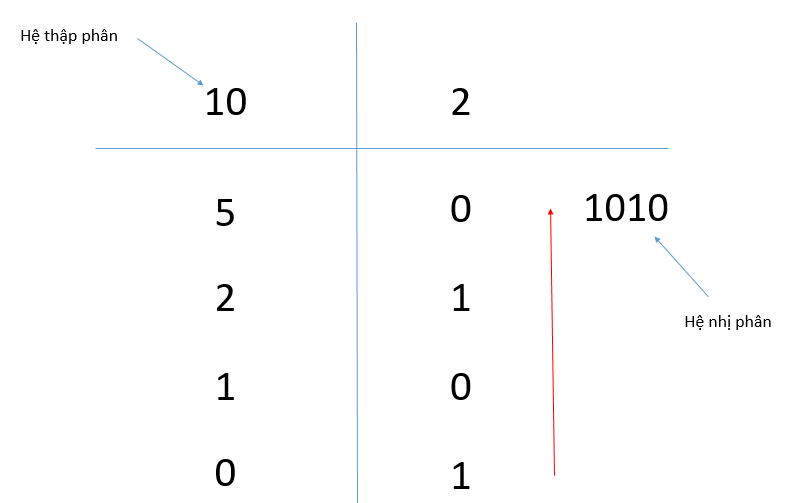
Để giải được bài toán, điều đầu tiên các bạn phải biết được số thập phân, nhị phân, bát phân, thập lục phân là gì? và cách chuyển đổi giữa các cơ chế này ra sao thì các bạn mới có thể làm được. Nếu các bạn chưa từng nghe các cơ số này thì có thể tham khảo trên google. Dưới đây mình sẽ hướng dẫn chi tiết cách chuyển đổi từ cơ số 10 sang cơ số 2, còn lại các cơ số khác tương tự nhé.

Hệ nhị phân là một hệ đếm dùng hai kí tự 0 và 1 để biểu đạt một số.

*Bài viết này được đăng tại [free tuts .net]*

Ví dụ: ta có một số nguyên 10 là số thập phân, khi chuyển sang số nhị phân sẽ là 1010.

Vậy làm sao nó có thể chuyển được như vậy, đơn giản chỉ là lấy số thập phân đó và chia lấy dư cho 2, khi đó các con số phần dư được lấy ngược từ dưới lên chính là hệ nhị phân.



Như các bạn đã biết thì Stack hoạt động theo quy tắc LIFO (last in first out), vậy tại sao chúng ta không sử dụng Stack vào bài toán này, thay vì ta lưu các số này vào mảng ta chỉ cần lưu nó vào Stack rồi lấy nó ra theo quy tắc LIFO là xong.

Cứ mỗi lần chia lấy dư như vậy ta sẽ lưu vào Stack, cho đến khi số chia bằng 0 thì ta thực hiện lấy phần tử đầu (top) trong Stack ra, như vậy dãy số được lấy ra chính là dãy nhị phân.

Tương tự như vậy, để chuyển sang cơ số 8 và 16 ta cũng thực hiện chia lấy dư.

**2. Ứng dụng Stack để chuyển đổi cơ số**

Trước khi bắt đầu viết hàm chuyển đổi cơ số, ta cần khởi tạo các cấu trúc Node và Stack, trong bài toán này mình sẽ sử dụng danh sách liên kết để cài đặt Stack.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | /\* khai báo Node \*/  struct node  {  **int** data;      struct node \*pNext;  };  typedef struct node NODE;    /\* khai báo cấu trúc của Stack \*/  struct stack  {      NODE \*pTop; // con trỏ quản lí đầu stack  };  typedef struct stack STACK;    /\* hàm khởi tạo Stack \*/  void KhoiTaoStack(STACK &s)  {      s.pTop = NULL;  }    /\* hàm khởi tạo 1 cái node \*/  NODE \*KhoiTaoNode(**int** x)  {    //tạo mới một NODE      NODE \*p = new NODE();      if (p == NULL)      {          cout << "\nKhông đủ bộ nhớ để cấp phát !";          return NULL;      }    // đưa dữ liệu của biến x vào trong cái data của node p      p->data = x;      p->pNext = NULL;      return p;  } |

Tiếp đến ta viết một hàm kiểm tra Stack rỗng, đây là điều kiện để có thể thêm phần tử hoặc xóa phần tử.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\* hàm kiểm tra Stack rỗng\*/  **bool** IsEmpty(STACK s)  {      // nếu stack rỗng      if (s.pTop == NULL)      {          return false;      }      return true;  } |

Để thêm được dữ liệu và Stack và lấy dữ liệu ra, ta cần có hàm push() và hàm pop().

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | /\* Thêm phần tử vào đầu Stack (top)\*/  **bool** Push(STACK &s, NODE \*p)  {      // node p đang rỗng      if (p == NULL)      {          return false;      }        // nếu danh sách rỗng      if (IsEmpty(s) == false)      {      // node p cũng chính là node pTop <=>chính là node đầu stack          s.pTop = p;      }      else      {      // B1: cho con trỏ của node p trỏ đến node pTop          p->pNext = s.pTop;      // B2: cập nhật lại node đầu chính là node p          s.pTop = p;      }    // thêm thành công      return true;  }    /\* Lấy phần tử đầu danh sách và hủy nó đi \*/  **bool** Pop(STACK &s, **int** &x) // x chính là giá trị cần lấy ra  {      // nếu danh sách rỗng      if (IsEmpty(s) == false)      {      // lấy thất bại <=> danh sách đang rỗng          return false;      }    // gán node đầu danh sách vào node p <=> node p chính là node mà tí nữa ta sẽ xóa nó      NODE \*p = s.pTop;     // cập nhật lại node đầu      s.pTop = s.pTop->pNext;    // lấy giá trị của node đầu ra gán vào biến x      x = p->data;    // lấy phần tử thành công      return true;  } |

Sau khi đã tạo xong các hàm cơ bản bây giờ chúng ta sẽ tạo hàm Chuyen\_Co\_So() để chuyển cơ số thập phân sang các cơ số khác. Tham số truyền vào của hàm bao gồm một Stack, cơ số cần đổi và số hệ thập phân cần chuyển.

Ta sẽ dùng vòng lặp while để chia lấy dư số cần chuyển đến khi số đó bằng 0, sau mỗi lần chia như vậy sẽ thêm vào Stack.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | /\* Hàm đổi cơ số 10 sang các cơ số 2, 8, 16 \*/  void Chuyen\_Co\_So(STACK &s, **int** cosocandoi, **int** hethapphan)  {      while (hethapphan != 0)      {  **int** x = hethapphan % cosocandoi;      // thêm x vào node p          NODE \*p = KhoiTaoNode(x);      // thêm node p vào stack          Push(s, p);      //tiếp tục chia đến hết          hethapphan /= cosocandoi;      }  } |

Sau khi chia xong chúng ta sẽ có một Stack bao gồm các số là phần dư. Nhiệm vụ của chúng ta bây giờ là tạo một hàm xuất các giá trị trong Stack ra theo cơ chế LIFO.

**\*Lưu ý:**Ở hệ bát phân và thập lục phân, khi giá trị lớn hơn 9 sẽ được quy ước thành các chữ cái, cụ thể như sau: 10 <=> A, 11 <=> B, 12 <=> C, 13 <=> D, 14 <=> E và 15 <=> F.

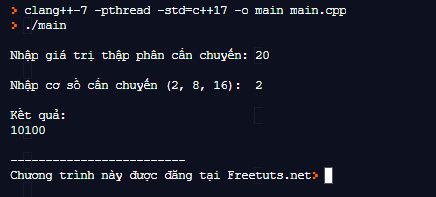
Vì vậy khi xuất các giá trị trong Stack ra màn hình ta cần thêm điều kiện, nếu giá trị < 10 thì ta thực hiện in bình thường, nếu giá trị >= 10 thì ta sẽ xuất ra các chữ cái quy ước.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | /\* xuất danh sách stack \*/  void XuatStack(STACK &s)  {      while (IsEmpty(s) == true)      {  **int** x;          Pop(s, x);      //nếu x < 10 thi xuất bình thường          if (x < 10)          {              cout << x;          }      //nếu x >= 10 thì xuất chữ cái tương ứng          else          {              if (x == 10)              {                  cout << "A";              }              else if (x == 11)              {                  cout << "B";              }              else if (x == 12)              {                  cout << "C";              }              else if (x == 13)              {                  cout << "D";              }              else if (x == 14)              {                  cout << "E";              }              else if (x == 15)              {                  cout << "F";              }          }      }  } |

**Full Code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193 | #include<iostream>  using namespace std;    /\*  Đổi 1 số nguyên sang cơ số 2 8 16  \*/    /\* khai báo Node \*/  struct node  {  **int** data;      struct node \*pNext;  };  typedef struct node NODE;    /\* khai báo cấu trúc của Stack \*/  struct stack  {      NODE \*pTop; // con trỏ quản lí đầu stack  };  typedef struct stack STACK;    /\* hàm khởi tạo Stack \*/  void KhoiTaoStack(STACK &s)  {      s.pTop = NULL;  }    /\* hàm khởi tạo 1 cái node \*/  NODE \*KhoiTaoNode(**int** x)  {    //tạo mới một NODE      NODE \*p = new NODE();      if (p == NULL)      {          cout << "\nKhông đủ bộ nhớ để cấp phát !";          return NULL;      }    // đưa dữ liệu của biến x vào trong cái data của node p      p->data = x;      p->pNext = NULL;      return p;  }    /\* hàm kiểm tra Stack rỗng\*/  **bool** IsEmpty(STACK s)  {      // nếu stack rỗng      if (s.pTop == NULL)      {          return false;      }      return true;  }    /\* Thêm phần tử vào đầu Stack (top)\*/  **bool** Push(STACK &s, NODE \*p)  {      // node p đang rỗng      if (p == NULL)      {          return false;      }        // nếu danh sách rỗng      if (IsEmpty(s) == false)      {      // node p cũng chính là node pTop <=>chính là node đầu stack          s.pTop = p;      }      else      {      // B1: cho con trỏ của node p trỏ đến node pTop          p->pNext = s.pTop;      // B2: cập nhật lại node đầu chính là node p          s.pTop = p;      }    // thêm thành công      return true;  }    /\* Lấy phần tử đầu danh sách và hủy nó đi \*/  **bool** Pop(STACK &s, **int** &x) // x chính là giá trị cần lấy ra  {      // nếu danh sách rỗng      if (IsEmpty(s) == false)      {      // lấy thất bại <=> danh sách đang rỗng          return false;      }    // gán node đầu danh sách vào node p <=> node p chính là node mà tí nữa ta sẽ xóa nó      NODE \*p = s.pTop;     // cập nhật lại node đầu      s.pTop = s.pTop->pNext;    // lấy giá trị của node đầu ra gán vào biến x      x = p->data;    // lấy phần tử thành công      return true;  }    /\* Xem thông tin của node đầu danh sách \*/  **bool** Top(STACK s, **int** &x)  // x chính là giá trị cần xem  {      // nếu danh sách rỗng        if (IsEmpty(s) == false)      {          return false;      }      x = s.pTop->data;      return true;  }    /\* Hàm đổi cơ số 10 sang các cơ số 2, 8, 16 \*/  void Chuyen\_Co\_So(STACK &s, **int** cosocandoi, **int** hethapphan)  {      while (hethapphan != 0)      {  **int** x = hethapphan % cosocandoi;      // thêm x vào node p          NODE \*p = KhoiTaoNode(x);      // thêm node p vào stack          Push(s, p);      //tiếp tục chia đến hết          hethapphan /= cosocandoi;      }  }    /\* xuất danh sách stack \*/  void XuatStack(STACK &s)  {      while (IsEmpty(s) == true)      {  **int** x;          Pop(s, x);      //nếu x < 10 thi xuất bình thường          if (x < 10)          {              cout << x;          }      //nếu x >= 10 thì xuất chữ cái tương ứng          else          {              if (x == 10)              {                  cout << "A";              }              else if (x == 11)              {                  cout << "B";              }              else if (x == 12)              {                  cout << "C";              }              else if (x == 13)              {                  cout << "D";              }              else if (x == 14)              {                  cout << "E";              }              else if (x == 15)              {                  cout << "F";              }          }      }  }    **int** main()  {      STACK s;      KhoiTaoStack(s);    **int** hethapphan,cosocandoi;      cout << "\nNhập giá trị thập phân cần chuyển: ";      cin >> hethapphan;    do{      cout << "\nNhập cơ số cần chuyển (2, 8, 16):  ";        cin >> cosocandoi;    }while(cosocandoi != 2 && cosocandoi != 8 && cosocandoi != 16);        Chuyen\_Co\_So(s, cosocandoi, hethapphan);      cout << "\nKết quả:\n";      XuatStack(s);      cout << endl;      cout<<"\n-------------------------\n";    cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**Mình chỉ thực hiện chuyển số thập phân sang nhị phân, các hệ khác các bạn có thể test thử nhé.



**3. Kết luận**

Như vậy là chúng ta đã thực hiện xong chương trình chuyển đổi cơ số thập phân sang cơ số nhị phân, bát phân, thập lục phân. Đây là một bài toán khá phổ biến trong các ngôn ngữ lập trình, vì vậy các bạn hãy nắm thật kỹ. Để có thể hiểu rõ hơn về Stack các bạn có thể luyện tập thêm nhiều bài toán khác áp dung Stack. Chúc các bạn thực hiện thành công !!!

**Bài tập kiểm tra số nguyên tố bằng Stack**

Trong hướng dẫn này mình sẽ thực hiện một chương trình nhập một dãy các số nguyên vào Stack sau đó thực hiện xuất các số nguyên tố ra màn hình. Đây là một bài tập khá đơn giản nhưng rất phổ biến trong lập trình.



Chúng ta sẽ cùng nhau tạo một cấu trúc Stack với danh sách liên kết, sau đó thực hiện xuất các số nguyên tố trong Stack ra màn hình.

**Mục lục**

* [1. Gợi ý cách thực hiện](https://freetuts.net/bai-tap-kiem-tra-so-nguyen-to-bang-stack-3174.html#goto-h2-0)
* [2. Chương trình xuất các số nguyên tố trong Stack](https://freetuts.net/bai-tap-kiem-tra-so-nguyen-to-bang-stack-3174.html#goto-h2-1)
* [3. Kết luận](https://freetuts.net/bai-tap-kiem-tra-so-nguyen-to-bang-stack-3174.html#goto-h2-2)

**1. Gợi ý cách thực hiện**

Trong chương trình này ta sẽ thực hiện nhập vào một dãy số nguyên sau đó lưu vào Stack, vậy nên việc đầu tiên ta cần tạo cấu trúc Stack. Trong hướng dẫn này mình sẽ sử dụng danh sách liên kết để cài đặt Stack, vì vậy ta cần tạo thêm cấu trúc Node.

Để có thể thêm, lấy các phần tử trong Stack thì ta cần tạo thêm các hàm cơ bản trong Stack như:

*Bài viết này được đăng tại [free tuts .net]*

* Hàm isEmpty.
* Hàm Push.
* Hàm Pop.

Khi này ta sẽ bắt đầu thực hiện tạo các hàm liên quan đến việc xuất các số nguyên tố trong Stack ra màn hình:

* Hàm getData() lấy dữ liệu từ người dùng sau đó đưa nó vào Stack
* Hàm ktSoNT() để kiểm tra các số trong Stack có phải là số nguyên tố hay không.
* Hàm XuatSoNguyenTo() để xuất các số nguyên tố ra màn hình.

**2. Chương trình xuất các số nguyên tố trong Stack**

Ta sẽ dựa vào gợi ý ở trên rồi lần lượt thực hiện tạo các hàm cần thiết cho bài toán, đầu tiên ta sẽ tạo cấu trúc Stack và cấu trúc Node trong danh sách liên kết để cài đặt Stack.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | /\* khai báo Node \*/  struct node  {  **int** data;      struct node \*pNext;  };  typedef struct node NODE;    /\* khai báo cấu trúc của Stack \*/  struct stack  {      NODE \*pTop; // con trỏ quản lí đầu stack  };  typedef struct stack STACK; |

Sau đó ta sẽ viết hàm khởi tạo Stack và hàm khởi tạo Node.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | /\* hàm khởi tạo Stack \*/  void KhoiTaoStack(STACK &s)  {      s.pTop = NULL;  }    /\* hàm khởi tạo 1 cái node \*/  NODE \*KhoiTaoNode(**int** x)  {    //tạo mới một NODE      NODE \*p = new NODE();      if (p == NULL)      {          cout << "\nKhông đủ bộ nhớ để cấp phát !";          return NULL;      }    // đưa dữ liệu của biến x vào trong cái data của node p      p->data = x;      p->pNext = NULL;      return p;  } |

Ta cần một hàm kiểm tra Stack có tồn tại phần tử hay không, đây là điều kiện để có thể thực hiện các thao tác khác trong Stack như thêm, lấy phần tử.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\* hàm kiểm tra Stack rỗng\*/  **bool** IsEmpty(STACK s)  {      // nếu stack rỗng      if (s.pTop == NULL)      {          return false;      }      return true;  } |

Và hai hàm cơ bản trong Stack đó chính là Push() và Pop(), đây là hai hàm không thể thiếu khi làm việc với Stack.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | /\* Thêm phần tử vào đầu Stack (top)\*/  **bool** Push(STACK &s, NODE \*p)  {      // node p đang rỗng      if (p == NULL)      {          return false;      }        // nếu danh sách rỗng      if (IsEmpty(s) == false)      {      // node p cũng chính là node pTop <=>chính là node đầu stack          s.pTop = p;      }      else      {      // B1: cho con trỏ của node p trỏ đến node pTop          p->pNext = s.pTop;      // B2: cập nhật lại node đầu chính là node p          s.pTop = p;      }    // thêm thành công      return true;  }    /\* Lấy phần tử đầu danh sách và hủy nó đi \*/  **bool** Pop(STACK &s, **int** &x) // x chính là giá trị cần lấy ra  {      // nếu danh sách rỗng      if (IsEmpty(s) == false)      {      // lấy thất bại <=> danh sách đang rỗng          return false;      }    // gán node đầu danh sách vào node p <=> node p chính là node mà tí nữa ta sẽ xóa nó      NODE \*p = s.pTop;     // cập nhật lại node đầu      s.pTop = s.pTop->pNext;    // lấy giá trị của node đầu ra gán vào biến x      x = p->data;    // lấy phần tử thành công      return true;  } |

Sau khi tạo các hàm cơ bản trong Stack, bây giờ ta sẽ bắt đầu tạo các hàm liên quan đến xuất các số nguyên tố. Đầu tiên sẽ là hàm kiểm tra số nguyên tố, đây là một thuật toán rất phô biến, vì vậy các bạn có thể tìm hiểu trên google hoặc có thể tham khảo của mình dưới đây.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | /\* hàm kiểm tra số nguyên tố \*/  **bool** ktSoNT(**int** n)  {      // Nếu n < 2 thì không phải là số nguyên tố      if (n < 2){          return false;      }      // ngược lại nếu n >= 2 thì ta xét điều kiện số nguyên tố      for (**int** i = 2; i < (n - 1); i++){        //nếu n chia hết cho i thì không phải là số nguyên tố          if (n % i == 0){              return false;          }      }       //ngược lại là số nguyên tố      return true;  } |

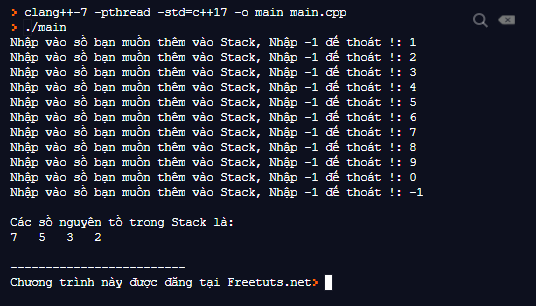
Và cuối cùng là hàm xuất các số là số nguyên tố trong Stack ra màn hình bằng cách sử dụng hàm ktSoNT() và hàm Pop(). Đơn giản chỉ là ta sẽ xét điều kiện nếu x (số thêm vào trong Stack) là số nguyên tố thì ta sẽ xuất ra màn hình.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | void XuatSoNguyenTo(STACK &s)  {      while (IsEmpty(s) == true)      {  **int** x;          Pop(s, x);      //nếu x là số nguyên tố thì ta xuất ra màn hình          if (ktSoNT(x))          {              cout << x << "\t";          }      }  } |

**Full Code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171 | #include<iostream>  using namespace std;    /\* khai báo Node \*/  struct node  {  **int** data;      struct node \*pNext;  };  typedef struct node NODE;    /\* khai báo cấu trúc của Stack \*/  struct stack  {      NODE \*pTop; // con trỏ quản lí đầu stack  };  typedef struct stack STACK;    /\* hàm khởi tạo Stack \*/  void KhoiTaoStack(STACK &s)  {      s.pTop = NULL;  }    /\* hàm khởi tạo 1 cái node \*/  NODE \*KhoiTaoNode(**int** x)  {    //tạo mới một NODE      NODE \*p = new NODE();      if (p == NULL)      {          cout << "\nKhông đủ bộ nhớ để cấp phát !";          return NULL;      }    // đưa dữ liệu của biến x vào trong cái data của node p      p->data = x;      p->pNext = NULL;      return p;  }    /\* hàm kiểm tra Stack rỗng\*/  **bool** IsEmpty(STACK s)  {      // nếu stack rỗng      if (s.pTop == NULL)      {          return false;      }      return true;  }    /\* Thêm phần tử vào đầu Stack (top)\*/  **bool** Push(STACK &s, NODE \*p)  {      // node p đang rỗng      if (p == NULL)      {          return false;      }        // nếu danh sách rỗng      if (IsEmpty(s) == false)      {      // node p cũng chính là node pTop <=>chính là node đầu stack          s.pTop = p;      }      else      {      // B1: cho con trỏ của node p trỏ đến node pTop          p->pNext = s.pTop;      // B2: cập nhật lại node đầu chính là node p          s.pTop = p;      }    // thêm thành công      return true;  }    /\* Lấy phần tử đầu danh sách và hủy nó đi \*/  **bool** Pop(STACK &s, **int** &x) // x chính là giá trị cần lấy ra  {      // nếu danh sách rỗng      if (IsEmpty(s) == false)      {      // lấy thất bại <=> danh sách đang rỗng          return false;      }    // gán node đầu danh sách vào node p <=> node p chính là node mà tí nữa ta sẽ xóa nó      NODE \*p = s.pTop;     // cập nhật lại node đầu      s.pTop = s.pTop->pNext;    // lấy giá trị của node đầu ra gán vào biến x      x = p->data;    // lấy phần tử thành công      return true;  }    /\* Xem thông tin của node đầu danh sách \*/  **bool** Top(STACK s, **int** &x)  // x chính là giá trị cần xem  {      // nếu danh sách rỗng        if (IsEmpty(s) == false)      {          return false;      }      x = s.pTop->data;      return true;  }  /\* hàm thêm dữ liệu vào Stack \*/  void getData(STACK &s,**int** x)  {      //tạo một node p để lưu giá trị của x vào          NODE \*p = KhoiTaoNode(x);      // thêm node p vào stack          Push(s, p);    }    /\* hàm kiểm tra số nguyên tố \*/  **bool** ktSoNT(**int** n)  {      // Nếu n < 2 thì không phải là số nguyên tố      if (n < 2){          return false;      }      // ngược lại nếu n >= 2 thì ta xét điều kiện số nguyên tố      for (**int** i = 2; i < (n - 1); i++){        //nếu n chia hết cho i thì không phải là số nguyên tố          if (n % i == 0){              return false;          }      }       //ngược lại là số nguyên tố      return true;  }  /\* hàm xuất các số nguyên tố ra màn hình \*/  void XuatSoNguyenTo(STACK &s)  {      while (IsEmpty(s) == true)      {  **int** x;          Pop(s, x);      //nếu x là số nguyên tố thì ta xuất ra màn hình          if (ktSoNT(x))          {              cout << x << "\t";          }      }  }    **int** main()  {    STACK s;    KhoiTaoStack(s);    **int** x;    while(1){      cout<<"Nhập vào số bạn muốn thêm vào Stack, Nhập -1 để thoát !: ";      cin >> x;      getData(s, x);      if(x == -1) break;    }      cout << "\nCác số nguyên tố trong Stack là: \n";    XuatSoNguyenTo(s);    cout << endl;      cout<<"\n-------------------------\n";    cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**

****

**3. Kết luận**

Như vậy là chúng ta đã thực hiện xong chương trình xuất các số là số nguyên tố trong Stack ra màn hình. Các bạn có thể luyện tập bằng cách xuất các số khác như số hoàn hảo, số chính phương, đây là một cách luyện tập rất hiệu quả cho thấy độ hiểu bài của các bạn. Chúc các bạn thực hiện thành công !!!