**Bài 7: Khái niệm về Stack - Sử dụng thư viện chuẩn STL cho C/C++**

**Đăng bởi: Admin | Lượt xem: 2319 | Chuyên mục: C/C++**

**Các tính năng cơ bản :**

Stack là một dạng chuyển đổi danh sách chứa, được thiết kế đặc biệt theo dạng LIFO (last-in first-out), các phần tử vào đầu tiên sẽ được lấy ra cuối cùng và ngược lại

Các ngăn xếp được triển khai như các bộ điều hợp vùng chứa, là các lớp sử dụng một đối tượng được đóng gói của một lớp chứa cụ thể làm vùng chứa bên dưới của nó, cung cấp một tập hợp các hàm thành viên cụ thể để truy cập các phần tử của nó. Các phần tử được đẩy  từ "phía sau" của danh sách chứa cụ thể, được gọi là đỉnh của ngăn xếp.

Container bên dưới có thể là bất kỳ mẫu lớp container tiêu chuẩn nào hoặc một số lớp container được thiết kế đặc biệt khác. Container sẽ hỗ trợ các hoạt động sau:

* empty
* size
* back
* push\_back
* pop\_back

  Tại mọi thời điểm, chúng ta duy trì một con trỏ tới phần tử dữ liệu vừa được PUSH cuối cùng vào trên ngăn xếp. Vì con trỏ này luôn biểu diễn vị trí trên cùng của ngăn xếp vì thế được đặt tên là **top**. **Con trỏ top** cung cấp cho chúng ta giá trị của phần tử trên cùng của ngăn xếp mà không cần phải thực hiện hoạt động xóa ở trên (hoạt động pop).

**Phương thức để hỗ trợ các tính năng của ngăn xếp.**

**Phương thức  peek() :**

Cấu trúc như sau :

Bắt đầu hàm peek

**return** stack[top]

kết thúc hàm

**int** **peek**() {

**return** stack[top];

}

**Phương thức isFull()**

Bắt đầu hàm isfull

**if** top bằng MAXSIZE

**return** true

**else**

**return** false

kết thúc **if**

kết thúc hàm

**bool** **isfull**() {

**if**(top == MAXSIZE)

**return** true;

**else**

**return** false;

}

**Phương thức isEmpty() :**

Giải thuật của hàm isEmpty():

bắt đầu hàm isempty

**if** top nhỏ hơn 1

**return** true

**else**

**return** false

kết thúc **if**

kết thúc hàm

Sự triển khai của hàm isEmpty() trong ngôn ngữ C khác hơn một chút. Chúng ta khởi tạo top tại -1, giống như chỉ mục của mảng bắt đầu từ 0. Vì thế chúng ta kiểm tra nếu top là dưới 0 hoặc -1 thì ngăn xếp là trống. Dưới đây là phần code:

**bool** **isempty**() {

**if**(top == -1)

**return** true;

**else**

**return** false;

}

**Hoạt động PUSH trong cấu trúc dữ liệu ngăn xếp**

Tiến trình đặt (thêm) một phần tử dữ liệu mới vào trên ngăn xếp còn được biết đến với tên Hoạt động PUSH. Hoạt động push bao gồm các bước sau:

* kiểm tra xem ngăn xếp đã đầy hay chưa.
* nếu ngăn xếp là đầy, tiến trình bị lỗi và thoát ra.
* nếu ngăn xếp chưa đầy, tăng top để trỏ tới phần bộ nhớ trống tiếp theo.
* thêm phần tử dữ liệu vào vị trí nơi mà top đang trỏ đến trên ngăn xếp.
* trả về success.

Diagram

Description automatically generated

Nếu Danh sách liên kết được sử dụng để triển khai ngăn xếp, thì ở bước 3 chúng ta cần cấp phát một không gian động.

Cách sử dụng PUSH như sau :

bắt đầu hoạt động push: stack, **data**

**if** stack là đầy

**return** null

kết thúc **if**

top ← top + 1

stack[top] ← **data**

kết thúc hàm

**void** **push**(**int** data) {

**if**(!isFull()) {

top = top + 1;

stack[top] = data;

}**else** {

printf("Khong the chen them du lieu vi Stack da day.\n");

}

}

**Hoạt động POP của cấu trúc dữ liệu ngăn xếp**

Việc truy cập nội dung phần tử trong khi xóa nó từ ngăn xếp còn được gọi là Hoạt động POP. Trong sự triển khai Mảng của hoạt động pop(), phần tử dữ liệu không thực sự bị xóa, thay vào đó **top** sẽ bị giảm về vị trí thấp hơn trong ngăn xếp để trỏ tới giá trị tiếp theo. Nhưng trong sự triển khai Danh sách liên kết, hoạt động pop() thực sụ xóa phần tử xữ liệu và xóa nó khỏi không gian bộ nhớ.

Hoạt động POP có thể bao gồm các bước sau:

* kiểm tra xem ngăn xếp là trống hay không.
* nếu ngăn xếp là trống, tiến trình bị lỗi và thoát ra.
* nếu ngăn xếp là không trống, truy cập phần tử dữ liệu tại **top** đang trỏ tới.
* giảm giá trị của top đi 1.
* trả về success.

Box and whisker chart

Description automatically generated with medium confidence

Cách sử dụng POP :

bắt đầu hàm pop: stack

**if** stack là trống

**return** null

kết thúc **if**

**data** ← stack[top]

top ← top - 1

**return** **data**

kết thúc hàm

**int** **pop**(**int** data) {

**if**(!isempty()) {

data = stack[top];

top = top - 1;

**return** data;

}**else** {

printf("Khong the lay du lieu, Stack la trong.\n");

}

}

**Ứng dụng của ngăn xếp :**

Xử lý gọi hàm trong C/C++ - Trong máy tính, sử dụng để tính giá trị biểu thức, xử lý ngắt - Trong các chương trình biên dịch - Trong trình duyệt web, trình soạn thảo văn bản - Định giá biểu thức + Biểu thức trung tố: toán tử hai ngôi đứng giưã hai toán hạng, toán tử một ngôi đứng trước toán hạng + Biểu thức hậu tố : toán tử đứng sau toán hạng + Biểu thức tiền tố  toán tử đứng trước toán hạng

VD định giá biểu thức A = b + c \* d /e – f Trung tố a\*(b-c)/d Hậu tố abc-\*d/ Tiền tố /\*a-bcd

**Bài 8: Các hàm cơ bản và bài tập minh họa về STACK - Sử dụng thư viện chuẩn STL cho C/C++**

**Đăng bởi: Admin | Lượt xem: 5194 | Chuyên mục: C/C++**

**Một số hàm quan trọng trong Stack :**

* Push : Thêm một phần tử vào đỉnh của ngăn xếp, số phần tử của ngăn xếp tăng lên 1.
* Pop: Xóa bỏ phần tử đầu tiên ở đỉnh của ngăn xếp, số phần tử của ngăn xếp giảm đi 1.
* Top: Lấy giá trị của phần tử đầu tiên ở đỉnh của ngăn xếp, số phần tử của ngăn xếp không thay đổi.
* IsEmpty: Kiểm tra ngăn xếp trống hay không. Ngăn xếp trống là ngăn xếp không có phần tử nào.
* IsFull: Kiểm tra ngăn xếp đã đầy hay chưa. Thao tác này không phải lúc nào cũng có.
* Size: Lấy số lượng phần tử stack đang có.

**Cú pháp sử dụng các hàm :**

**Kiểm tra stack đầy(IsFull) :**

Hàm này sẽ kiểm tra xem *stack*hiện tại đã đầy hay chưa. Nếu chỉ số top của *stack*đang bằng với capacity - 1, tức stack đã đầy.

**bool** **IsFull**(**int** capacity){

**if**(top >= capacity - 1){

**return** true;

}**else**{

**return** false;

}

}

**Kiểm tra stack rỗng(IsEmpty)**

Nếu như stack đang không có phần tử nào, ta sẽ gán chỉ số top = -1 để đánh dấu. Như vậy, để kiểm tra stack có đang rỗng hay không rất đơn giản. Ta chỉ cần so sánh giá trị top có phải -1 hay không mà thôi.

**bool** **IsEmpty**(){

**if**(top == -1){

**return** true;

}**else**{

**return** false;

}

}

**Thêm phần tử vào đỉnh stack(Push) :**

Chúng ta sẽ chỉ có thể push(thêm phần tử) vào đỉnh stack khi stack chưa đầy. Nếu stack đầy, chúng ta sẽ đưa ra thông báo và không thực hiện push. Ngược lại, ta sẽ tăng top lên một đơn vị và gán giá trị cho phần tử tại chỉ số top.

**void** **Push**(**int** stack[], **int** value, **int** capacity){

**if**(IsFull(capacity) == true){

printf("\nStack is full. Overflow condition!");

}**else**{

++top;

stack[top] = value;

}

}

**Xóa phần tử khỏi đỉnh stack(Pop)**

Chúng ta sẽ chỉ có thể pop(xóa phần tử) khỏi đỉnh stack khi stack không trống. Nếu stack trống, chúng ta sẽ đưa ra thông báo và không thực hiện pop. Ngược lại, ta sẽ giảm giá trị top đi một đơn vị.

**void** **Pop**(){

**if**(IsEmpty() == true){

printf("\nStack is empty. Underflow condition!");

}**else**{

--top;

}

}

**Lấy giá trị phần tử ở đỉnh stack(Top) :**

Để lấy giá trị phần tử ở đỉnh stack, ta có thao tác rất đơn giản:

**int** **Top**(**int** stack[]){

**return** stack[top];

}

**Lấy số lượng phần tử stack đang có(Size)**

Biến *top*lưu chỉ số lớn nhất của stack. Như vậy, việc lấy size của stack cực kỳ đơn giản:

**int** **Size**(){

**return** top + 1;

}

Và cuối cùng, chúng ta sẽ có 1 chương trình cài đặt stack hoàn thiện như sau:

#**include** <stdio.h>

**int** top = -1;

**bool** **IsFull**(**int** capacity){

**if**(top >= capacity - 1){

**return** true;

}**else**{

**return** false;

}

}

**bool** **IsEmpty**(){

**if**(top == -1){

**return** true;

}**else**{

**return** false;

}

}

**void** **Push**(**int** stack[], **int** value, **int** capacity){

**if**(IsFull(capacity) == true){

printf("\nStack is full. Overflow condition!");

}**else**{

++top;

stack[top] = value;

}

}

**void** **Pop**(){

**if**(IsEmpty() == true){

printf("\nStack is empty. Underflow condition!");

}**else**{

--top;

}

}

**int** **Top**(**int** stack[]){

**return** stack[top];

}

**int** **Size**(){

**return** top + 1;

}

**int** **main**(){

**int** capacity = 3;

**int** top = -1;

**int** stack[capacity];

// pushing element 5 in the stack .

Push(stack, 5, capacity);

printf("\nCurrent size of stack is %d", Size());

Push(stack, 10, capacity);

Push(stack, 24, capacity);

printf("\nCurrent size of stack is %d", Size());

// As the stack is full, further pushing will show an overflow condition.

Push(stack, 12, capacity);

//Accessing the top element

printf("\nThe current top element in stack is %d", Top(stack));

//Removing all the elements from the stack

**for**(**int** i = 0 ; i < 3;i++)

Pop();

printf("\nCurrent size of stack is %d", Size());

//As the stack is empty , further popping will show an underflow condition.

Pop();

}

**Bài tập ứng dụng :**

Bài 1 : chuyển từ thập phân sang nhị phân dùng stack trong :

#**include** <stdio.h>

#**include** <conio.h>

#**include** <memory.h>

#**include** <malloc.h>

#**include** <time.h>

**class** **Stack**

{

**public**:

Stack(**void**);

Stack(**int** size);

**virtual** ~Stack(**void**);

**int** **Push**(**int** number);

**int** **Pop**(**int**& number);

**int** **SetSize**(**const** **int** size);

**void** **Clear**();

**int** **GetNumberElement**() **const**;

**private**:

Stack& **operator**=(**const** Stack& stack);

Stack(**const** Stack& stack);

**int** m\_index;

**int** m\_size;

**int**\* m\_data;

};

Stack::Stack(**void**)

{

m\_size = 0;

m\_data = 0;

m\_index = -1;

}

Stack::Stack(**int** size)

{

**if**(size > 0)

{

m\_size = size;

m\_data = (**int** \*)malloc(size \* **sizeof**(**int**));

memset(m\_data, 0, size \* **sizeof**(**int**));

}

**else**

{

m\_size = 0;

m\_data = 0;

}

m\_index = -1;

}

Stack::~Stack(**void**)

{

Clear();

}

**int** **Stack::SetSize**(**const** **int** size)

{

**if**(size > 0)

{

**if**(m\_data)

{

free(m\_data);

}

m\_index = -1;

m\_size = size;

m\_data = (**int** \*)malloc(size \* **sizeof**(**int**));

memset(m\_data, 0, size \* **sizeof**(**int**));

}

**return** 0;

}

**int** **Stack::Push**(**int** number)

{

**if**(m\_index < m\_size)

{

m\_data[++m\_index] = number;

**return** 1;

}

**return** 0;

}

**int** **Stack::Pop**(**int**& number)

{

**if**(m\_index >= 0)

{

number = m\_data[m\_index--];

**return** true;

}

**return** false;

}

**void** **Stack::Clear**()

{

**if**(m\_data)

free(m\_data);

m\_data = 0;

m\_size = 0;

m\_index = -1;

}

**int** **Stack::GetNumberElement**() **const**

{

**return** m\_index + 1;

}

**int** **main**(**int** argc, **char**\* argv[])

{

**int** number;

printf("n = ");

scanf("%d", &number);

**int** k = number;

Stack **my\_stack**(200);

**while**(number)

{

my\_stack.Push(number % 2);

number /= 2;

}

printf("\nConvert to binary:\n");

printf("%d(10) = ", k);

**while**(my\_stack.GetNumberElement())

{

**int** number;

my\_stack.Pop(number);

printf("%d", number);

}

printf("(2)\n");

printf("\nDone!!!!!\n");

getch();

**return** 0;

}