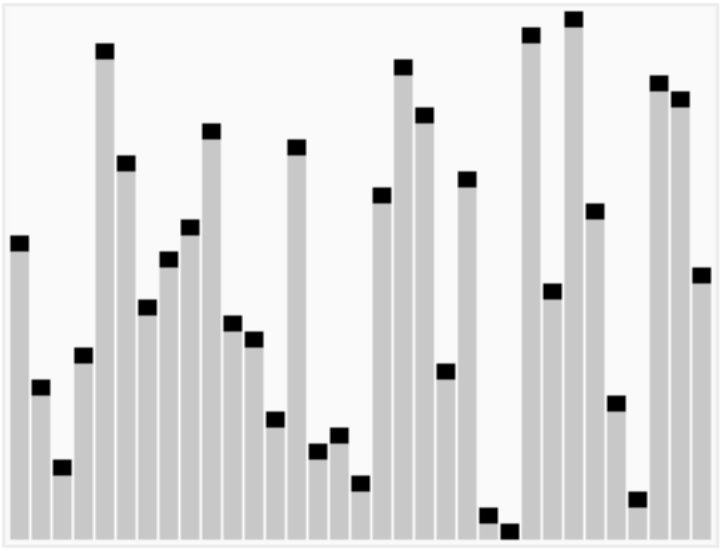
QUICK SORT

Ý tưởng: QuickSort chia mảng thành hai danh sách bằng cách so sánh từng phần tử của danh sách với một phần tử được chọn được gọi là phần tử chốt. Những phần tử nhỏ hơn hoặc bằng phần tử chốt được đưa về phía trước và nằm trong danh sách con thứ nhất, các phần tử lớn hơn chốt được đưa về phía sau và thuộc danh sách con thứ hai. Cứ tiếp tục chia như vậy tới khi các danh sách con đều có độ dài bằng 1

– Chọn phần tử đứng đầu hoặc đứng cuối làm phần tử chốt.
– Chọn phần tử đứng giữa danh sách làm phần tử chốt.
– Chọn phần tử trung vị trong 3 phần tử đứng đầu, đứng giữa và đứng cuối làm phần tử chốt.
– Chọn phần tử ngẫu nhiên làm phần tử chốt (Cách này hay được chọn vì tránh được các trường hợp đặc biệt)
Hình minh họa:



Cài đặt bằng C:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>      // thu vien de khoi tao tham so cho ham rand()
#define swap(type, a, b) {type temp = a; a = b; b = temp; } // hang hoan vi

void quickSort(int *a, int l, int r)
{
    srand(time(NULL));  //khoi tao tham so ham rand()
    int key = a[l + rand() % (r-l+1)];  //lay khoa la gia tri ngau nhien tu a[l] -> a[r]
    //int key = a[(l+r)/2];
    int i = l, j = r;

    while(i <= j)
    {
        while(a[i] < key) i++;        // tim phan tu ben trai ma >=key
        while(a[j] > key) j--;        // tim phan tu ben trai ma <=key
        if(i <= j)
        {
            if (i < j)
                swap(int, a[i], a[j]);  // doi cho 2 phan tu kieu int a[i], a[j].
```

```
            i++;
            j--;
        }
    }
    //bay gio ta co 1 mang : a[l]....a[j]..a[i]...a[r]
    if (l < j) quickSort(a, l, j);   // lam lai voi mang a[l]....a[j]
    if (i < r) quickSort(a, i, r); // lam lai voi mang a[i]....a[r]
}

int main ()
{
    int i, arr[] = { 40, 10, 100, 90, 20, 25 };
    quickSort(arr, 0, 5);
    for (i=0; i<6; i++)
        printf ("%d ", arr[i]);
    return 0;
}
```

Thuật toán được cài đặt từ dòng 6 đến dòng 29. Trong đó có dòng 21 **swap(int, a[i], a[j]);** là một hằng dùng để hoán vị 2 số a[i] và a[j] được định nghĩa tại dòng 4 và nó tương đương với đoạn code sau (bạn có thể thay nó trực tiếp vào):

```
    {
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
```

Bạn cũng có thể xem đoạn code sau, với đoạn code này tôi đã in ra màn hình mọi sự biến đổi của mảng, của l, r, i, j giúp chúng ra quan sát được chính xác sự thay đổi trong từng bước:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>      // thu vien de khoi tao tham so cho ham rand()
#define swap(type, a, b) {type temp = a; a = b; b = temp; } // hang hoan vi

void quickSort(int *a, int l, int r)
{
    srand(time(NULL));  //khoi tao tham so ham rand()
    int key = a[l + rand() % (r-l+1)];  //lay khoa la gia tri ngau nhien tu a[l] -> a[r]
    //int key = a[(l+r)/2];
    int i = l, j = r;
    printf("\n\nl = %d   r = %d    select: key = %d  (random) ",l, r, key);

    while(i <= j)
    {
        //printf("n");
        printf("\n\n\ti : %d", i);
        while (a[i] < key) { i++; printf(" -> %d",i); }
        printf("\n\tj : %d", j);
        while (a[j] > key) { j--; printf(" -> %d",j); }
        if(i <= j)
        {
            if (i < j)
            {
                swap(int, a[i], a[j]);  // doi cho 2 phan tu kieu int a[i], a[j].
                printf("\n\tswap(a[%d], a[%d])  swap(%d, %d)", i, j, a[i], a[j]);
            }
            i++;
            j--;
            printf("\n\tarr[] : ");
            for (int i=0; i<8; i++)
                printf ("%d ", a[i]);
```

```
        }
    }
    //bay gio ta co 1 mang : a[l]....a[j]..a[i]...a[r]
    if (l < j) quickSort(a, l, j);   // lam lai voi mang a[l]....a[j]
    if (i < r) quickSort(a, i, r); // lam lai voi mang a[i]....a[r]
}

int main ()
{
    int i, arr[] = { 2, 8, 7, 1, 3, 5, 6, 4  };
    int n = 8;
    printf("\n\nArray befor sort: ");
    for (i=0; i<n; i++)
        printf ("%d ", arr[i]);

    quickSort(arr, 0, n-1);

    printf("\n\nArray after sort: ");
    for (i=0; i<n; i++)
        printf ("%d ", arr[i]);
}
```

Ta nhận thấy hiệu quả của thuật toán phụ thuộc vào việc chọn giá trị mốc (hay phần tử chốt).

– Trường hợp tốt nhất: mỗi lần phân hoạch ta đều chọn được phần tử median (phần tử lớn hơn hay bằng nửa số phần tử và nhỏ hơn hay bằng nửa số phần tử còn lại) làm mốc. Khi đó dãy được phân hoạch thành hai phần bằng nhau, và ta cần log2(n) lần phân hoạch thì sắp xếp xong. Ta cũng dễ nhận thấy trong mỗi lần phân hoạch ta cần duyệt qua n phần tử. Vậy độ phức tạp trong trường hợp tốt nhất thuộc O(nlog2(n)).
– Trường hợp xấu nhất: mỗi lần phần hoạch ta chọn phải phần tử có giá trị cực đại hoặc cực tiểu làm mốc. Khi đó dãy bị phân hoạch thành hai phần không đều: một phần chỉ có một phần tử, phần còn lại có n-1 phần tử. Do đó, ta cần tới n lần phân hoạch mới sắp xếp xong. Vậy độ phức tạp trong trường hợp xấu nhất thuộc O(n2).

*** Vậy ta có độ phức tạp của Quick Sort như sau:**
– Trường hợp tốt nhất: O(nlog2n)
– Trường hợp xấu nhất: O(n2)
– Trường hợp trung bình: O(nlog2n)

Khử đệ quy trong Quic Sort

Bản chất của các giải thuật đệ quy là lưu trữ các tham biến đệ quy vào một ngăn xếp (stack) để lần lượt lấy ra xử lý. Như vậy nếu gặp dữ liệu lớn sẽ dễ gây tràn stack. Khi khử đệ quy của giải thuật đệ quy, với mỗi lần cần gọi hàm quicksort trong đệ quy ta thay bằng cách lưu lại các giá trị bên trái và bên phải của 2 dãy con vào 2 stack Sl và Sr, khi nào cần sẽ gọi ra. Ngoài ra chúng ta cũng có thể lưu chung các giá trị bên trái và bên phải vào 1 Stack, khi lấy ra sẽ lấy 2 phần tử liên tiếp.

b1: Khởi tạo 2 Stack rỗng Sl, Sr để lưu các giá trị l và các giá trị r của mỗi mảng con.
b2: Ban đầu dãy đang xét từ 0 đến n-1, ta lưu l vào Sl và r vào Sr
b3: Lấy l ra từ Sl, r ra từ Sr
b4: Phân hoạch dãy A[L] .. A[R] thành 2 dãy A[l]..A[j] và A[i]..A[r]
b5: Nếu l < j lưu l và j vào Sl và Sr, Nếu i < r lưu i và r vào Sl và Sr,
b6: Nếu Stack không rỗng quay lại b3.

Sau đây là code cài đặt, trong code có dùng đến stack trong C++, nếu bạn không muốn hoặc chưa quen dùng stack trong C++ có thể tham khảo cách xây dựng 1 Stack luôn.

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

```
#include<time.h>      // thu vien de khoi tao tham so cho ham rand()
#include<stack>       // Su dung Stack trong C++
#define swap(type, a, b) {type temp = a; a = b; b = temp; }

using namespace std;

void quickSortUnRecursive(int *a, int l, int r)
{
    srand(time(NULL));
    stack Sl;       // Stack left
    stack Sr;       // Stack right
    Sl.push(l);         // dua phan tu dau tien l vao Sl
    Sr.push(r);         // dua phan tu dau tien r vao Sl

    while(!Sl.empty())  // trong khi stack khong rong
    {

        int l = Sl.top(); Sl.pop();     // lay phan tu dau tien trong Sl l ra
        int r = Sr.top(); Sr.pop();     // lay phan tu dau tien trong Sr r ra
        int key = a[l + rand() % (r-l+1)];
        int i = l, j = r;
        while (i <= j)       // phan hoach thanh 2 day con
        {
            while (a[i] < key) i++;
            while (a[j] > key) j--;

            if(i <= j)
            {
                if (i < j)
                    swap(int, a[i], a[j]);
                i++;
                j--;
            }
        }
        if (l < j) { Sl.push(l); Sr.push(j); }   // dua 2 dau mut l va j vao Sl va
        if (i < r) { Sl.push(i); Sr.push(r); }   // dua 2 dau mut i va r vao Sl va
    }
}

int main ()
{
    int i, arr[] = { 40, 10, 100, 90, 20, 25 };
    quickSortUnRecursive(arr, 0, 5);
    for (i=0; i<6; i++)
        printf ("%d ", arr[i]);
    return 0;
}
```

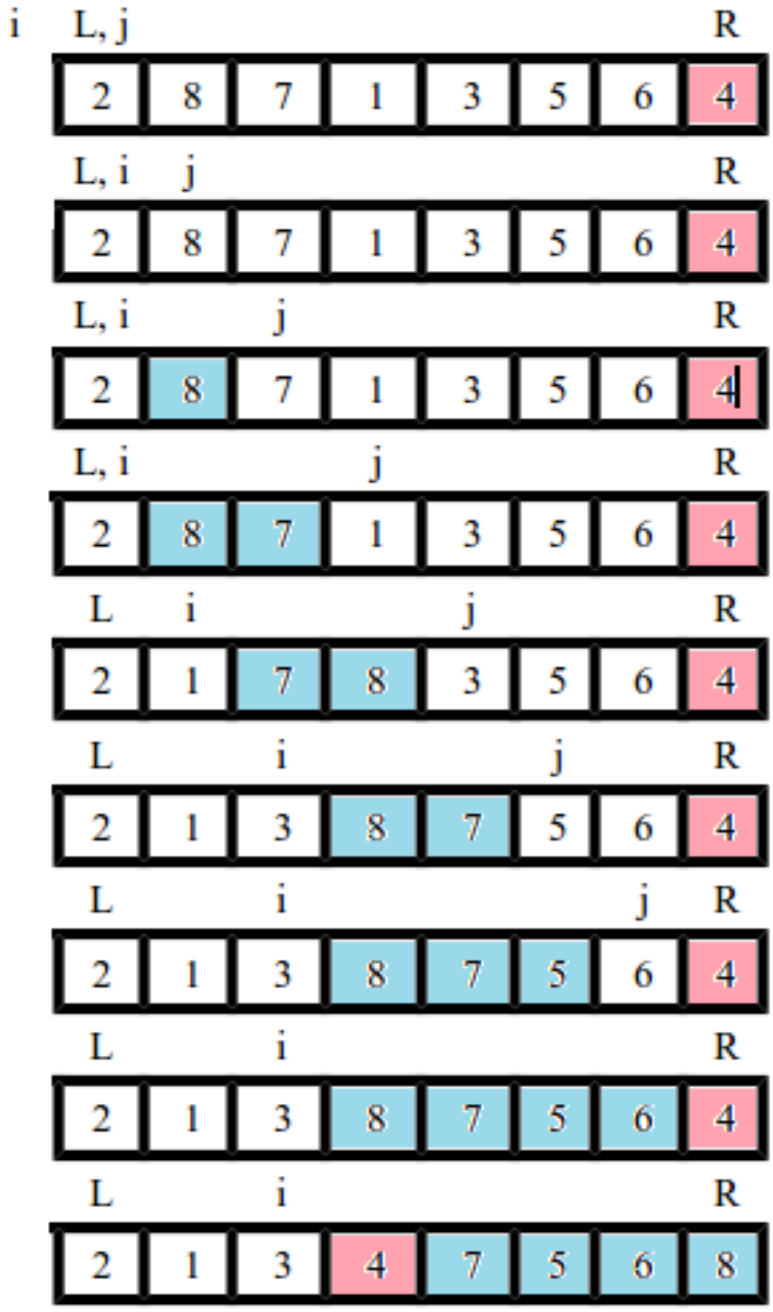
Sử dụng đệ quy đuôi trong Quick Sort

Ta cũng có thể sử dụng thuật toán đệ quy đuôi để xây dựng thuật toán Quick Sort, tức là chỉ dùng một lời gọi đệ quy trong hàm, tuy nhiên để thực hiện việc này chúng ta sẽ phải chọn phần tử chốt là phần tử đầu tiên hoặc cuối cùng của mảng cần sắp xếp. Ta Có thể phân tích trường hợp này như sau:

Trước tiên ta xác định việc phân dãy ra làm 2 mảng con như sau:

```
int partition(int *a, int l, int r)
{
    int key = a[r];     // xac dinh khoa
```

```
    int i = l - 1, j;   // khoi tao i, j
    for (j=l; j<r; j++)  // duyet mang
        if (a[j] <= key) //neu a[j] <= key
        {
            i++;
            swap(int, a[i], a[j]);  // hoan vi
        }
    swap(int, a[i+1], a[r]);    // hoan vi voi phan tu khoa
    return i + 1;   // tra ve vi tri cua khoa
}
```



Sau khi phân chia dãy ta được 2 dãy, dãy bên trái là những phần tử nhỏ hơn hoặc bằng key, bên phải là dãy các phần tử lớn hơn key.

Bây giờ ta xây dựng Quick Sort đệ quy đuôi như sau:

```
void TailRecursiveQuicksort(int *a, int l, int r)
```

```
{
    while (l<r)
    {
        int q = partition(a,l,r);   //lay vi tri khoa
        TailRecursiveQuicksort(a, l, q-1);      // sap xep day ben trai
        l = q + 1;      // khi day ben trai sap xep xong ta chuyen sang day ben phai
    }
}
```

Ngoài ra, nhằm mục đích sao cho khi gọi đệ quy, Stack của chúng ta có thể phải chứa càng ít càng tốt, vì vậy ta cải tiến một chút giải thuật như sau:

```
void TailRecursiveQuicksort_2(int *a, int l, int r)
{
    while (l<r)
    {
        int q = partition(a,l,r);
        if (q < (l + (r-l)/2))   //Neu vi tri khoa < 1/2 do dai day thi sap xep ben
        {
            TailRecursiveQuicksort_2(a, l, q-1);
            l = q + 1;
        }
        else  //Neu vi tri khoa >= 1/2 do dai day thi sap xep ben trai
        {
            TailRecursiveQuicksort_2(a, q + 1, r);
            r = q-1;
        }
    }
}
```

Sử dụng Quick Sort xây dựng sẵn trong C++

Trong thư viện của C++ đã xây dựng sẵn cho chúng ta hàm qsort và chúng ta có thể sử dụng ngay nó.

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

int arr[] = { 40, 10, 100, 90, 20, 25 };

int compare (const void * a, const void * b)
{
  return ( *(int*)a > *(int*)b );
}

int main ()
{
      int n;
      qsort (arr + 1, 4, sizeof(int), compare);
      for (n=0; n<6; n++)
         printf ("%d ",arr[n]);
      return 0;
}
```

Trong lời gọi hàm qsort: **qsort (arr + 1, 4, sizeof(int), compare);** **arr + 1** chính là vị trí mảng bắt đầu sắp xếp, ở đây tương đương với sắp xếp từ phần tử a[1] = 10 và **sắp xếp 4 phần tử** tức là từ a[1] đến a[4]. **sizeof(int)** là kích thước 1 phần tử trong mảng. Nếu mảng là kiểu thực ta sẽ có là sizeof(float). **compare** được xây dựng với cấu trúc như sau dùng để xác định mảng được sắp xếp tăng hay giảm

```
int compareMyType (const void * a, const void * b)
{
```

```
    return ( *(MyType*)a @ *(MyType*)b );
}
```

Trong đó MyType là kiểu của các phần tử mảng. “@” là một phép toán nào đó được định nghĩa. Trong thư viện đã định nghĩa sẵn cho chúng ta 3 phép toán >, ==, <. Trong ví dụ này ta dùng *(int*)a > *(int*)b để sắp xếp tăng, ngược lại nếu *(int*)b > *(int*)a sẽ sắp xếp giảm.

Sắp xếp mảng cấu trúc bằng Quick Sort có sẵn trong C++

Tiếp theo chúng ta sẽ làm thế nào nếu chúng ta có 1 mảng các phần tử cấu trúc Struct và muốn sắp xếp theo 1 trường nào đó.

Ta sẽ định nghĩa 1 phép toán “@” để so sánh 1 trường nào đó bằng hàm **bool operator** sau đó xây dựng lại hàm **int compare** thep phép toán đã được xây dựng:

```
#include<iostream>
#include<cstdlib>
using namespace std;

struct St
{
    string name;
    int scores;
};

bool operator * ( const St &t, const St &u )    // dinh nghia phep toan "*"
{
    return t.name > u.name;  //sap xep tang dan theo ten
}

bool operator/( const St &t, const St &u )  // dinh nghia phep toan "/"
{
    return t.scores < u.scores;  //sap xep giam dan theo diem
}

int compare_name (const void * a, const void * b)
{
    return ( *( St*)a * *(St*)b );  //chi duoc su dung phep toan "*" de sap xep
}

int compare_scores (const void * a, const void * b)
{
    return ( *( St*)a / *(St*)b );  //chi duoc su dung phep toan "/" de sap xep
}

int main ()
{
    St arr[] = { {"Hien", 9}, {"Cuong", 7}, {"Nhung", 8}, {"Nam", 6} };

    int n = sizeof( arr ) / sizeof( St );
    qsort (arr, n, sizeof(St), compare_name);

    cout << "\nDanh sach xap xep tang dan theo ten:\n";
    for (int i = 0; i < n; i++ )
        cout << arr[ i ].name << ' ' << arr[ i ].scores << endl;

    qsort (arr, n, sizeof(St), compare_scores);
    cout << "\nDanh sach xap xep giam dan theo diem:\n";
```

```
        for (int i = 0; i < n; i++ )
            cout << arr[ i ].name << ' ' << arr[ i ].scores << endl;

        return 0;
    }
```

Trong đoạn code trên tôi đã dùng các ký hiệu đặc biệt * và / (mà chính xác hơn là các ký hiệu bình thường được dùng làm phép toán nhân và chia) để định nghĩa phép toán cho cách sắp xếp. Chúng ta có thể dùng một số ký tự phép toán khác như %, ^, *, /, +, -, >, =, < để định nghĩa phép toán mới.

HEAP SORT

. Stack cài đặt trên mảng

code by nguyenvanquan7826

```
#define Max 100 //so phan tu toi da cua Stack
typedef int item; //kieu du lieu cua Stack
struct Stack
{
    int Top; //Dinh Top
    item Data[Max]; //Mang cac phan tu
};
```

1.1 Khởi tạo danh sách rỗng, kiểm tra danh sách rỗng, đầy

code by nguyenvanquan7826

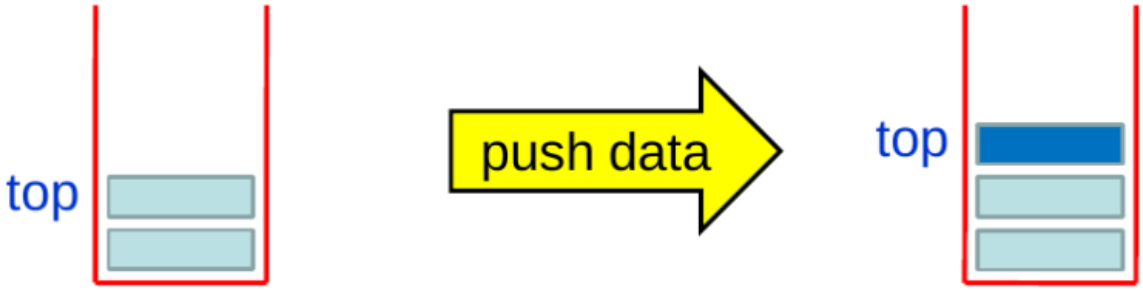
```
void Init (Stack &S) //khoi tao Stack rong
{
    S.Top = 0; //Stack rong khi Top la 0
}

int Isempty(Stack S) //kiem tra Stack rong
{
    return (S.Top == 0);
}

int Isfull(Stack S) //kiem tra Stack day
{
    return (S.Top == Max); //
}
```

1.2 Thêm phần tử vào Stack (Push)

Để chèn thêm phần tử vào Stack ta chèn vào vị trí Top, và tang Top lên 1 đơn vị



code by nguyenvanquan7826

```
void Push(Stack &S, item x) //them phan tu vao Stack
{
    if (!Isfull(S))
    {
```

```
            S.Data[S.Top] = x; //Gan du lieu
            S.Top ++; //Tang Top len 1
        }
    }
```

1.3 Lấy dữ liệu tại Top nhưng không xóa (Peak)

code by nguyenvanquan7826

```
int Peak(Stack S) //Lay phan tu o dau Stack nhung khong xoa
{
    return S.Data[S.Top-1]; //Lay du lieu tai Top
}
```

1.4 Xóa và lấy dữ liệu tại Top (Pop)

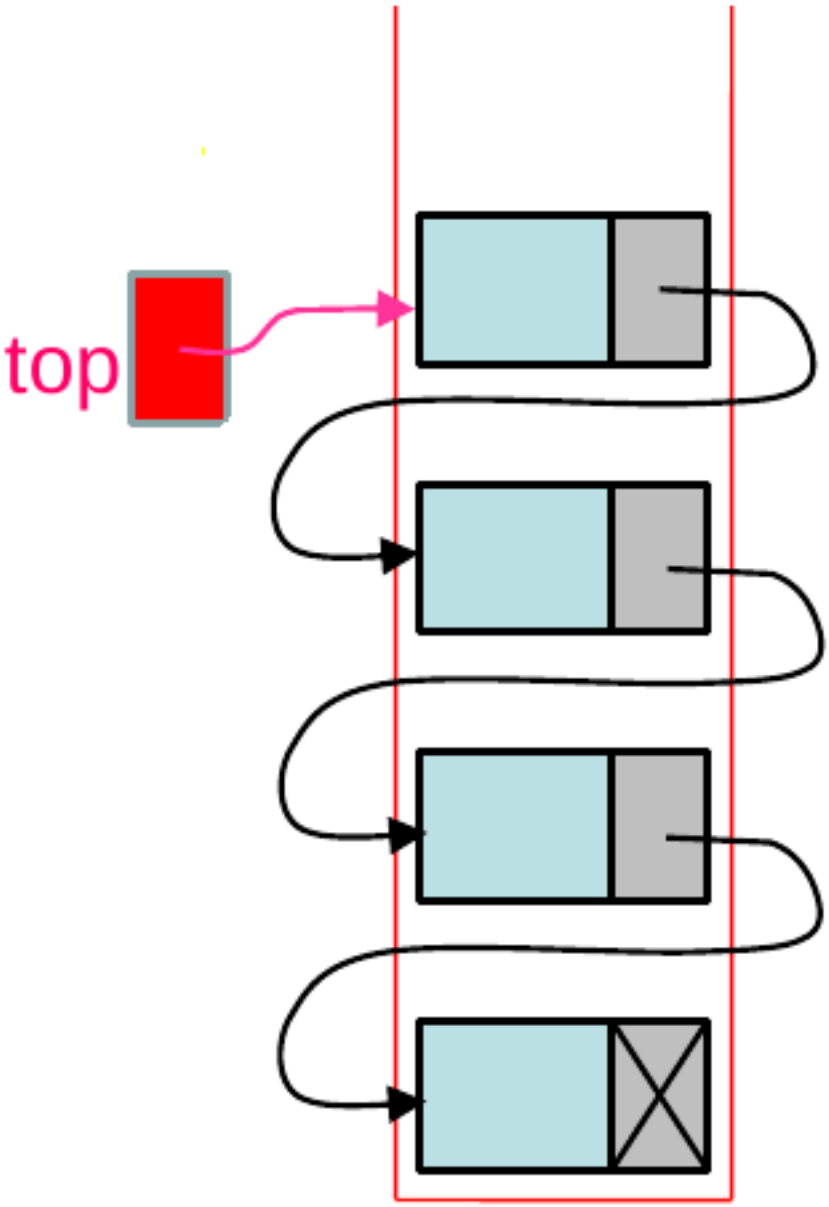
code by nguyenvanquan7826

```
    int Pop(Stack &S) //Loai bo phan tu khoi Stack
    {
        if (!Isempty(S))
        {
            S.Top --; //Giam Top
            return S.Data[S.Top]; //Lay du lieu tai Top
        }
    }
```

1.5 Code toàn bộ chương trình (full)

2. Stack cài đặt trên con trỏ

Stack trên con trỏ vẫn là stack bình thường nhưng link hoạt hơn vì nó dùng con trỏ để cấp phát và quản lý, không bị thừa hoặc thiếu gì cả.



2.1 Xây dựng cấu trúc

code by nguyenvanquan7826

```
typedef int item; //kieu du lieu
struct Node
{
    item Data; //du lieu
    Node *Next; //link
};
typedef struct Stack
{
    Node *Top;
};
```

2.2 Khởi tạo danh sách rỗng, kiểm tra danh sách rỗng, độ dài danh sách

```
void Init (Stack &S) //khoi tao Stack rong
{
    S.Top = NULL;
}

int Isempty(Stack S) //kiem tra Stack rong
{
    return (S.Top == NULL);
}

int Len (Stack S)
{
    Node *P = S.Top;
    int i=0;
    while (P != NULL) //trong khi chua het Stack thi van duyet
    {
        i++;
        P = P->Next;
    }
    return i;
}
```

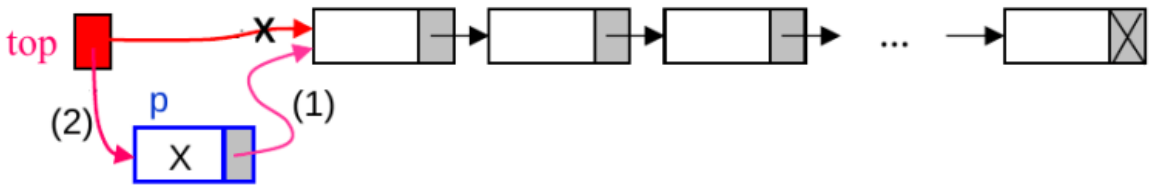
2.3 Tạo 1 Node

code by nguyenvanquan7826

```
Node *MakeNode(item x) //tao 1 Node
{
    Node *P = (Node*) malloc(sizeof(Node));
    P->Next = NULL;
    P->Data = x;
    return P;
}
```

2.4 Chèn phần tử vào Stack (Push)

Để chèn phần tử vào Stack thì chỉ cần cho con trỏ Note đó trỏ và Top, rồi Top trỏ lại nó là xong



code by nguyenvanquan7826

```
void Push(Stack &S, item x) //them phan tu vao Stack
{
    Node *P = MakeNode(x);
    P->Next = S.Top;
    S.Top = P;
}
```

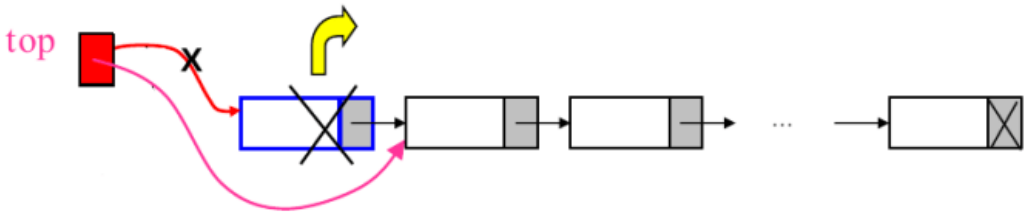
2.5 Lấy dữ liệu tại Top nhưng không xóa (Peak)

code by nguyenvanquan7826

```
    int Peak(Stack S) //Lay phan tu o dau Stack nhung khong xoa
    {
        return S.Top->Data;
    }
```

2.6 Xóa và lấy dữ liệu tại Top (Pop)

Ta chỉ cần cho con trỏ Top trỏ đến vị trí thứ 2 thôi.



code by nguyenvanquan7826

```
int Pop(Stack &S) //Loai bo phan tu khoi Stack
{
    if (!Isempty(S))
    {
        item x = S.Top->Data; //luu lai gia tri
        S.Top = S.Top->Next; //Xoa phan tu Top
        return x;
    }
}
```

2.7 Chương trình hoàn chỉnh (full code)

3. Sử dụng Stack có sẵn trong C++

Trong C++ đã xây dựng sẵn các phương thức (hàm) liên quan đến Stack, ta chỉ việc khai báo và sử dụng

code by nguyenvanquan7826

```
#include <iostream> // io
#include <stack> // use stack

using namespace std;

int main() {
    stack <int> S; // khai bao Stack voi kieu du lieu la int

    for (int i = 0; i < 10; i++) {
        S.push(i*78 + 26); // chen cac phan tu vao stack
    }

    cout << "Do dai stack: " << S.size() << "\n";

    // xuat tat ca phan tu trong stack
    while (!S.empty()) {    // trong khi stack chua trong
        int x = S.top();    // lay gia tri top
        S.pop();            // loai bo khoi stack
        cout << x << "  ";
    }
    cout << "\n";

    return 0;
}
```

4. VÍ dụ về ứng dụng của Stack

Stack có nhiều ứng dụng, sau đây là 1 ứng dụng trong chuyển đổi cơ số. Code sau sẽ chuyển số cơ số 10 sang cơ số x nhập từ bàn phím

code by nguyenvanquan7826

```
#include <iostream> // io
#include <stack> // use stack
```

```
using namespace std;

int main() {
    char num[] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
    stack <char> S; // khai bao Stack voi kieu du lieu la int

    int coSo, so, du;

    cout << "Nhap so can chuyen: ";
    cin >> so;

    cout << "Nhap co so can chuyen: ";
    cin >> coSo;

    // chuyen doi bang cach dua vao Stack
    while(so) {
        du = so % coSo;
        S.push(num[du]);
        so /= coSo;
    }

    // Xuat ra
    while(!S.empty()) {
        cout << S.top();
        S.pop();
    }

    return 0;
}
```

#LTTD

Group : https://www.facebook.com/groups/1014672612051349