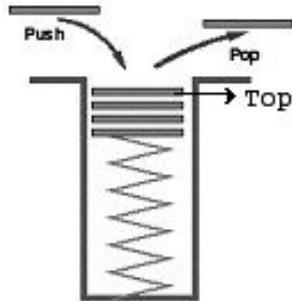


تراكيب البيانات Data Structure (1)

المكدس Stack.

المكدس يعتبر طريقة تخزين وإسترجاع بيانات ، حيث أن المُخزّن أخيراً يُسترجع أولاً "LIFO Last In First Out".



والمكدس له عدة تطبيقات منها معالجة أولوية التعابير الجبرية " + - * / " ، وتخزين عناوين رجوع الدوال عند تنفيذها.

والدوال الرئيسية للمكدس هي :

POP لإخراج عنصر من المكدس

Push لوضع عنصر في المكدس

والبعض إضافة دوال فرعية أخرى مثل:

IsEmpty وتستخدم للكشف على المكدس هل هو خالي أم لا

IsFull وتستخدم للكشف على المكدس هل هو مملوء أم لا

ويجب تحديد نوع المكدس الذي يكون هو نفسه نوع البيانات المراد تخزينها داخله. ومؤشر يشير الى قمة المكدس ويسمى بمؤشر المكدس Stack Pointer.

مثال: الـ Stack بإستخدام المصفوفات Arrays

```
#include <iostream.h>
#define Size 10

char Stack[Size];
int StackPointer;

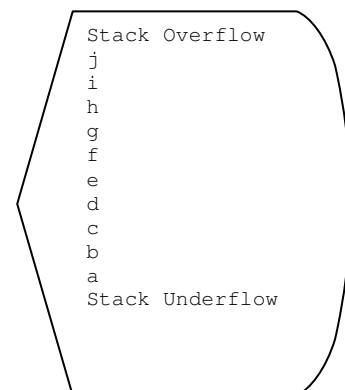
void Push(char c)
{
    if(StackPointer<Size-1)
        Stack[++StackPointer]=c;
    else
        cout << "Stack Overflow" << endl;
}

char Pop()
{
    if(StackPointer<0){
        cout << "Stack Underflow" << endl;
        return NULL;
    }
    else
        return Stack[StackPointer--];
}

void main()
{
    StackPointer=-1;
    Push('a');      Push('b');      Push('c');
    Push('d');      Push('e');      Push('f');
    Push('g');      Push('h');      Push('i');
    Push('j');
    Push('k');      // Stack Overflow

    cout << Pop() << endl;
    cout << Pop() << endl;
    cout << Pop() << endl;
    cout << Pop() << endl;
    cout << Pop() << endl;
    cout << Pop() << endl;
    cout << Pop() << endl;
    cout << Pop() << endl;
    cout << Pop() << endl;

    cout << Pop() << endl;    // Stack Underflow
}
```



مثال: الـ Stack باستخدام المؤشرات Pointers

```
#include <iostream.h>

struct Node{
    char Data;
    Node* Next;
};

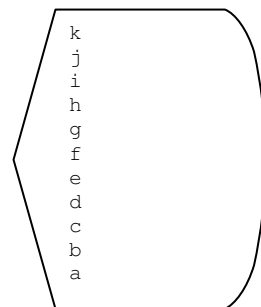
Node* Head;

bool Empty()
{
    return (Head==NULL);
}

void Push(char c)
{
    Node* p;
    p=new Node;
    p->Data=c;
    p->Next=Head;
    Head=p;
}

char Pop()
{
    char c;
    Node* p=Head;
    if(p){
        c=p->Data;
        Head=Head->Next;
        delete p;
    }
    else {
        cout << "Stack Underflow" << endl;
        c= NULL;
    }
    return c;
}

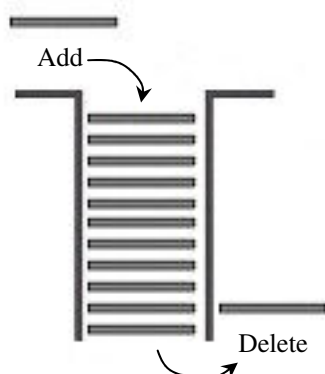
void main()
{
    Head=NULL;
    Push('a');    Push('b');    Push('c');
    Push('d');    Push('e');    Push('f');
    Push('g');    Push('h');    Push('i');
    Push('j');    Push('k');
    while (!Empty())
        cout << Pop() << endl;
}
```



مثال 2

الطابور Queue.

الطابور أيضاً يعتبر طريقة تخزين وإسترجاع بيانات ، حيث أن المُخزّن أولاً يُسترجع أولاً "FIFO First In First Out".



والطابور له عدة تطبيقات أغلبها له علاقة بأولوية المعالجة.

والدوال الرئيسية للطابور هي :

Add لإدخال عنصر في الطابور

Delete لإخراج عنصر من الطابور

والبعض يفضل استخدام دوال فرعية أخرى مثل:

IsEmpty وتستخدم للكشف على الطابور هل هو خالي أم لا.

وللطاير مؤشرين أحدهما لمقدمة الطاير ويسمى Front والآخر للمؤخرة ويسمى Rear.
مثال: ال Queue بإستخدام المؤشرات Pointers

```
#include <iostream.h>

struct Node{
    char Data;
    Node* Next;
};

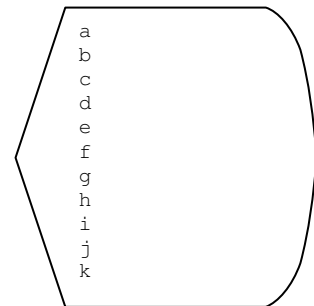
Node *Front, *Rear;

bool Empty()
{
    return (Front==NULL);
}

void Add(char c)
{
    Node* p;
    p=new Node;
    p->Data=c;
    p->Next=NULL;
    if (Front==NULL)
        Front=Rear=p;
    else{
        Rear->Next=p;
        Rear=Rear->Next;
    }
}

char Delete()
{
    char c;
    Node* p=Front;
    if(p){
        c=p->Data;
        Front=Front->Next;
        delete p;
    }
    else {
        cout << "Empty Queue..." << endl;
        c= NULL;
    }
    return c;
}

void main()
{
    Front=Rear=NULL;
    Add('a');      Add('b');      Add('c');
    Add('d');      Add('e');      Add('f');
    Add('g');      Add('h');      Add('i');
    Add('j');      Add('k');
    while (!Empty())
        cout << Delete() << endl;
}
```



مثال 3

السلاسل Linked Lists

السلاسل عبارة عن طريقة تخزين بيانات في الذاكرة ، ولا يمكن تمثيل السلاسل إلا عن طريق المؤشرات. وميزة السلاسل أنها تقدم أفضل حل بالنسبة لحجز اماكن في الذاكرة ، إذ أننا نستطيع حجز وإلغاء هذه الأماكن وقت تشغيل البرنامج Run-time.

تختص جميع السلاسل بالخصائص التالية:
1- كلها لها Head يشير الى بداية السلسلة.

- 2- كل خلية تحتوي على نوع البيانات المراد تخزينه ، ومؤشر يشير الى الخلية التي تليها.
- 3- آخر خلية في السلسلة يشير الى مكان محدد في الذاكرة يسمى NULL.

طريقة "LIFO" لبناء السلسلة:

1- أول خلية:

- a. حجز مكان للخلية في الذاكرة.
- b. تخزين البيانات في الخلية.
- c. وضع NULL في مؤشر الخلية.
- d. تغيير Head ليشير لهذه الخلية.

2- باقي الخلايا:

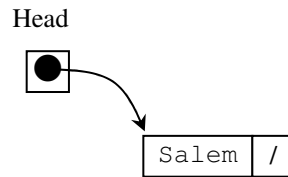
- a. حجز مكان للخلية في الذاكرة.
- b. تخزين البيانات في الخلية.
- c. ندع مؤشر الخلية يوضح الى Head.
- d. تغيير Head ليشير لهذه الخلية.

الرسومات الآتية توضح خطوات المثال رقم 4.

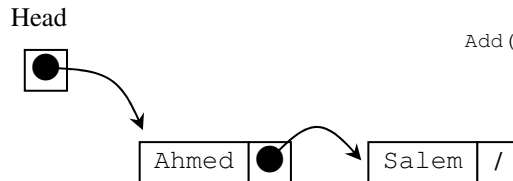
الخطوة 1: Head=NULL;



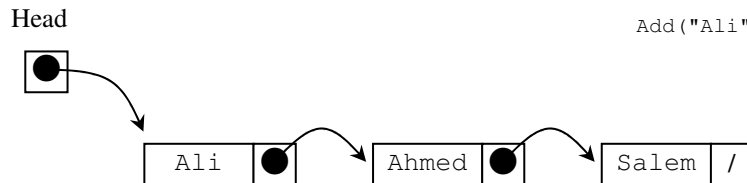
الخطوة 2: Add("Salem");



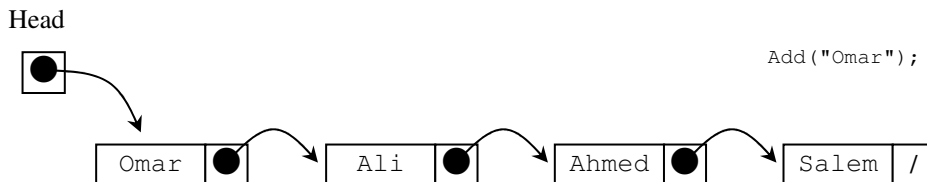
الخطوة 3: Add("Ahmed");



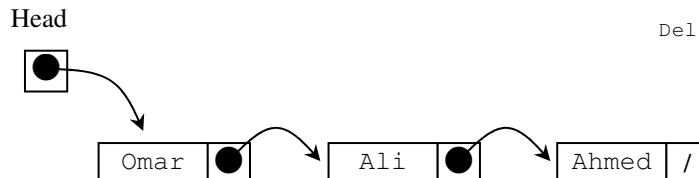
الخطوة 4: Add("Ali");



الخطوة 5: Add("Omar");



الخطوة 6: Del("Salem");



```

#include <iostream.h>
#include <string.h>

struct Node{
    char Data[10];
    Node* Next;};
Node* Head;

void Add(char* c)
{
    Node* p;
    p=new Node;
    strcpy(p->Data,c);
    p->Next=Head;
    Head=p;
}

void Del(char* c)
{
    Node* p=Head, *q;
    if(!strcmp(p->Data,c)) {
        Head=Head->Next;
        delete p;
    }
    else{
        while(p) {
            q=p;
            p=p->Next;
            if(!strcmp(p->Data,c)) {
                q->Next=p->Next;
                delete p;
                p=NULL;
            }
        }
    }
}

void PrintList()
{
    Node* p=Head;
    while(p){
        cout<< p->Data<<" ";
        p=p->Next;
    }
    cout << endl;
}

void main()
{
    Head=NULL; //1
    Add("Salem");PrintList(); //2
    Add("Ahmed");PrintList(); //3
    Add("Ali");    PrintList(); //4
    Add("Omar");PrintList(); //5
    Del("Salem");  PrintList(); //6
}

```

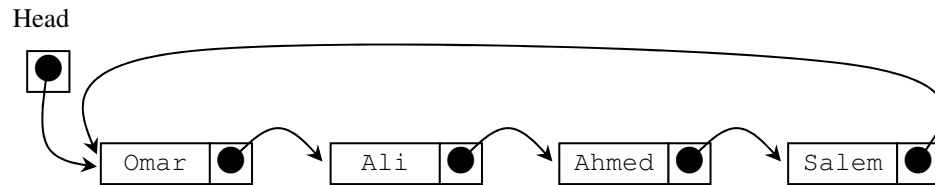
مثال 4

Salem
 Ahmed Salem
 Ali Ahmed Salem
 Omar Ali Ahmed Salem
 Omar Ali Ahmed

الدالة *strcpy(dst,src)* المعرفة في الملف *string.h* تقوم بنسخ قيمة *src* ووضعها في *dst* ، حيث كل من *dst* ، *src* من نوع *char**. والدالة *strcmp(str1,str2)* المعرفة في نفس الملف تقارن بين كل من *str1*،*str2* وترجع 0 إن كانا متساويين.

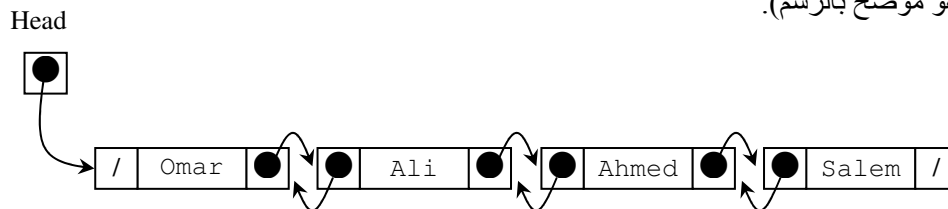
السلسلة الدائرية Circular Linked Lists

وهي السلسلة التي يكون مؤشر آخر خلية يشير الى بدايتها.(كما هو موضح بالرسم).



السلسلة المزدوجة Doubly Linked List

وهي السلسلة التي يوجد فيها مؤشرين بحيث يؤشر أحدهما الى الخلية التالية والآخر الى الخلية السابقة.(كما هو موضح بالرسم).



ويكون تعريفها كالتالي:

```
struct Node{
    char Data[10];
    Node* Next;
    Node* Prev;
};
```

الدوال التي تنفذ نفسها Recursion Functions

العديد من المسائل يمكن حلها بهذا النوع من الدوال مثل المضروب وأرقام Fibonacci. وسنتعرف لاحقاً على طرق زيارة الأشجار الثنائية (Binary Tree traversing) باستخدام هذا النوع من الدوال.

```
int fact( int n )
{
    if ( n == 0 ) return 1;
    else
        return n*fact(n-1);
}
```

مثال 5

المثال رقم 5 يوضح استخدام Recursion Function في حساب المضروب.

الترتيب Sorting

بالنظر الى مخرجات أي برنامج ستجد أنها ، عادة، مرتبة بشكل ما. ولهذا تعتبر مهمة ترتيب البيانات من أحد أكثر المهام المستخدمة من قبل المبرمجين. توجد عدة طرق لترتيب البيانات سنتطرق الى عدد منها، ولتوضيح الفرق بين الطرق سنوحد البيانات في جميع الأمثلة والتي ستكون عبارة عن مصفوفة معرفة بالشكل التالي:

```
int a[]={77,44,99,66,33,55,88,22,44};
```

اما الدالة Swap فعملها هو تبديل قيم المتغيرين a,b.

```
void swap(int& a,int& b)
{
    int t;
    t=a;
    a=b;
    b=t;
}
```

طريقة Bubble Sort

```
/* Bubble sort for integers */
void bubble( int a[], int n )
{
    int i, j;
    for(i=1;i<n;i++)
        for(j=1;j<=n-i;j++)
            if( a[j-1]>a[j]) swap(a[j-1],a[j]);
}
```

مثال 6

0	j-1	j	77	44	99	66	33	55	88	22	44
1	0	1	44	77	99	66	33	55	88	22	44
2	2	3	44	77	66	99	33	55	88	22	44
3	3	4	44	77	66	33	99	55	88	22	44
4	4	5	44	77	66	33	55	99	88	22	44
5	5	6	44	77	66	33	55	88	99	22	44
6	6	7	44	77	66	33	55	88	22	99	44
7	7	8	44	77	66	33	55	88	22	44	99
8	1	2	44	66	77	33	55	88	22	44	99
9	2	3	44	66	33	77	55	88	22	44	99
10	3	4	44	66	33	55	77	88	22	44	99
11	5	6	44	66	33	55	77	22	88	44	99
12	6	7	44	66	33	55	77	22	44	88	99
13	1	2	44	33	66	55	77	22	44	88	99
14	2	3	44	33	55	66	77	22	44	88	99
15	4	5	44	33	55	66	22	77	44	88	99
16	5	6	44	33	55	66	22	44	77	88	99
17	0	1	33	44	55	66	22	44	77	88	99
18	3	4	33	44	55	22	66	44	77	88	99
19	4	5	33	44	55	22	44	66	77	88	99
20	2	3	33	44	22	55	44	66	77	88	99
21	3	4	33	44	22	44	55	66	77	88	99
22	1	2	33	22	44	44	55	66	77	88	99
23	0	1	22	33	44	44	55	66	77	88	99

بنتبع الدالة في المثال رقم 6 والتي تمثل **Bubble Sort** نجد أنه تم إجراء 23 عملية تبديل في الذاكرة لترتيب مصفوفة حجمها 9 ، وأن عدد المقارنات التي تم إجراؤها لترتيبها هو 36 والذي حصلنا عليه من :
 8 مقارنات عندما $i=1$ ، 7 مقارنات عندما $i=2$ ، 6 مقارنات عندما $i=3$ وهكذا
 عدد المقارنات $= 1+2+3+4+5+6+7+8 = 36$
 ولحساب عدد المقارنات لمصفوفة بحجم n فإن :
 عدد المقارنات $= 1+2+.....+(n-3)+(n-2)+(n-1)$
 ومجموع هذه السلسلة الحسابية هو $n(n-1)/2$ وكلما كبرت قيمة n كلما اعتبر الواحد المطروح من
 n قيمة صغيرة يمكن أهملها ... فيبقى لدينا $n*n/2$ ولهذا تصنف طريقة الترتيب هذه على أنها من الدرجة
 $O(n^2)$ وذلك للعلاقة الطردية بين n^2 وعدد المقارنات.

طريقة Selection Sort

```
/* selection sort for integers */
void SelectionSort( int a[], int n )
{
    for(int i=0;i<n-1;i++)
    {
        int min=i;
        for(int j=i+1;j<n;j++)
            if( a[j]<a[min]) min=j;
        swap(a[min],a[i]);
    }
}
```

مثال 7

0	min	i	77	44	99	66	33	55	88	22	44
1	7	0	22	44	99	66	33	55	88	77	44
2	4	1	22	33	99	66	44	55	88	77	44
3	4	2	22	33	44	66	99	55	88	77	44
4	8	3	22	33	44	44	99	55	88	77	66
5	5	4	22	33	44	44	55	99	88	77	66
6	8	5	22	33	44	44	55	66	88	77	99
7	7	6	22	33	44	44	55	66	77	88	99
8	7	7	22	33	44	44	55	66	77	88	99

بنتبع الدالة في المثال رقم 7 والتي تمثل **Selection Sort** نجد أنه تم إجراء 8 عمليات تبديل في الذاكرة لترتيب مصفوفة حجمها 9 ، وأن عدد المقارنات التي تم إجراؤها لترتيبها هو 36. وتصنف طريقة الترتيب هذه على أنها من الدرجة $O(n^2)$ وذلك للعلاقة الطردية بين n^2 وعدد المقارنات.

طريقة Insertion Sort

```
/* Selection sort for integers */
void InsertionSort( int a[], int n )
{
    for(int i=1;i<n;i++)
    {
        int temp=a[i];
        for(int j=i;j>0 && a[j-1]>temp;j--)
            a[j]=a[j-1];
        a[j]=temp;
    }
}
```

مثال 8

0	i	77	44	99	66	33	55	88	22	44
1	1	44	77	99	66	33	55	88	22	44
2	2	44	66	77	99	33	55	88	22	44
3	3	33	44	66	77	99	55	88	22	44
4	4	33	44	55	66	77	99	88	22	44
5	5	33	44	55	66	77	88	99	22	44
6	6	22	33	44	55	66	77	88	99	44
7	7	22	33	44	44	55	66	77	88	99

بنتبع الدالة في المثال رقم 8 نجد أن فكرة عمل **Insertion Sort** تكمن في حشر القيمة الأصغر وأزاحة القيم الأخرى. أيضاً نجد أنه تم إجراء 30 عملية تبديل في الذاكرة لترتيب مصفوفة حجمها 9، وأن عدد المقارنات التي تم إجراؤها لترتيبها تقريباً 36. وتصنف طريقة الترتيب هذه على أنها من الدرجة $O(n^2)$ وذلك للعلاقة الطردية بين n^2 وعدد المقارنات.

Quick Sort طريقة

```
int partition( int *a, int low, int high ) {
    int left, right;
    int pivot_item;
    pivot_item = a[low];
    pivot = left = low;
    right = high;
    while ( left < right ) {
        /* Move left while item < pivot */
        while( a[left] <= pivot_item ) left++;
        /* Move right while item > pivot */
        while( a[right] >= pivot_item ) right--;
        if ( left < right ) swap(a[left],a[right]);
    }
    /* right is final position for the pivot */
    a[low] = a[right];
    a[right] = pivot_item;
    return right;
}

quicksort( void *a, int low, int high )
{
    int pivot;
    /* Termination condition! */
    if ( high > low )
    {
        pivot = partition( a, low, high );
        quicksort( a, low, pivot-1 );
        quicksort( a, pivot+1, high );
    }
}
```

مثال 9

0	77	44	99	66	33	55	88	22	44
1	22	44	99	66	33	55	88	77	44
2	22	44	33	66	99	55	88	77	44
3	22	44	33	44	99	55	88	77	66
4	22	33	44	44	99	55	88	77	66
5	22	33	44	44	55	99	88	77	66
6	22	33	44	44	55	66	88	77	99
7	22	33	44	44	55	66	77	88	99

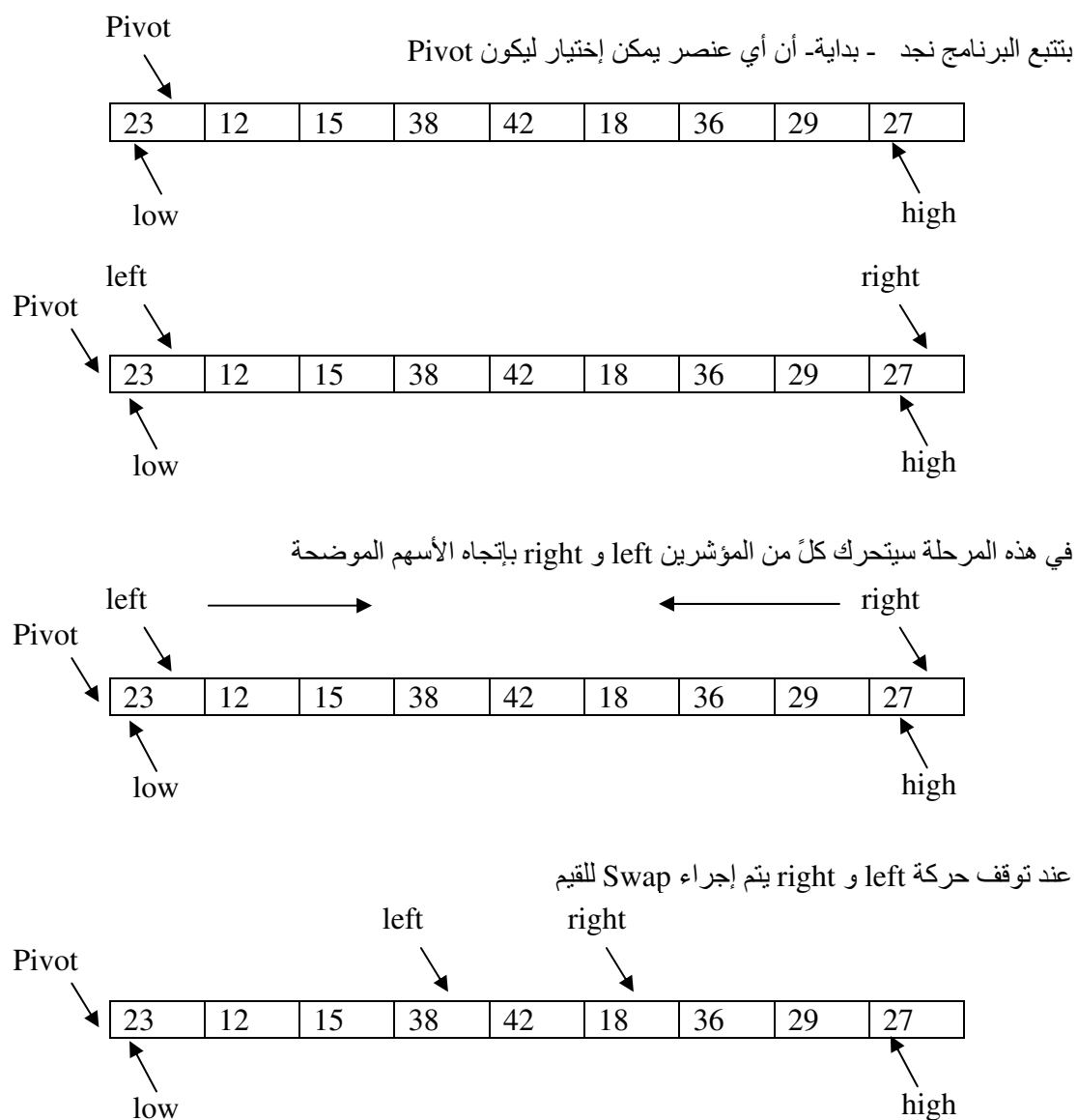
طريقة Quick Sort هي الأكثر كفاءة من بين كل طرق الترتيب المعروفة ، وهي تمثل تطبيق عملي للخوارزميات المبنية على أساس قاعدة "فرق تسد" Divide and Conquer ، والتي تظهر بوضوح في جزئي الخوارزمية في المثال 9.

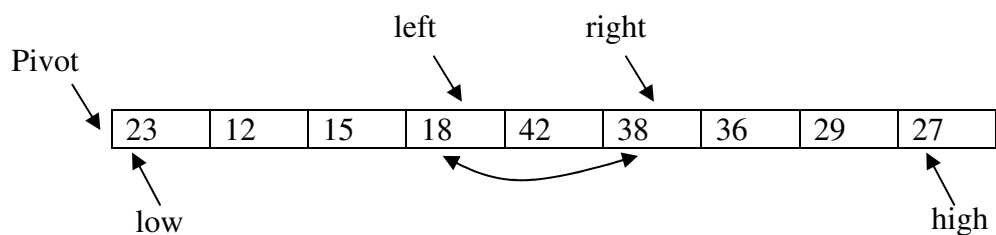
وتكمن فكرة البرنامج في تقسيم المصفوفة الى جزئين يقسمهم عنصر يسمى بـ Pivot بحيث :

< pivot	pivot	> pivot
---------	-------	---------

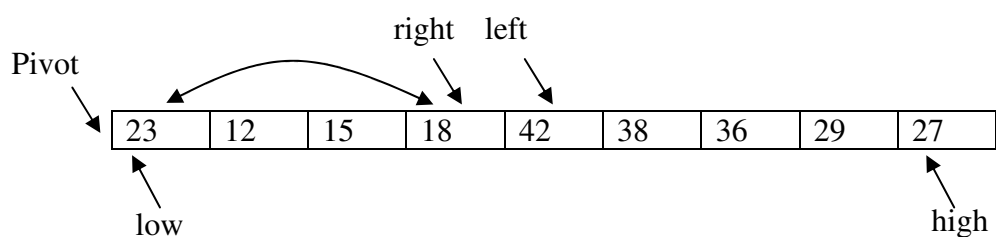
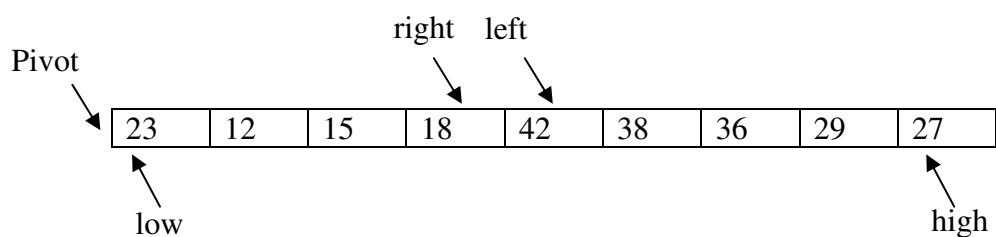
ويتكرر نفس العمل مع كل جزء كالتالي

< pivot				> pivot		
< p'	p'	> p'	pivot	< p''	p''	> p''

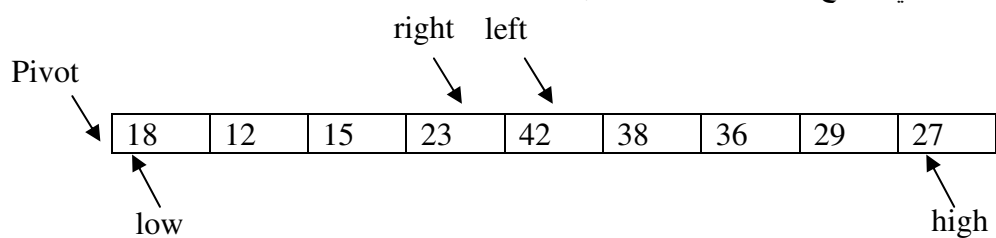




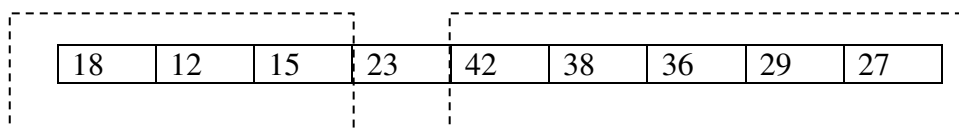
وتستمر حركة left و right حتى يتحقق الشرط `while ( left < right )`



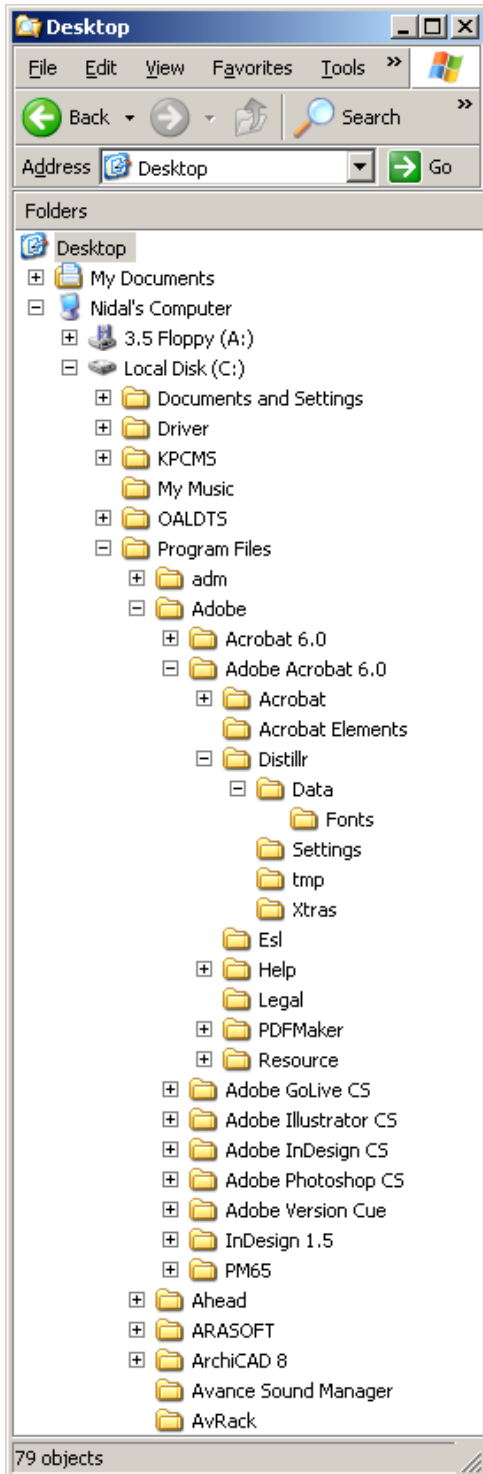
وهذه آخر خطوة وهي ترجيع مكان Pivot الجديد `return right;`



وهذا مكان Pivot الجديد سيكون هو نقطة تقسيم المصفوفة الى جزئين ، ومنها يتم اختيار Pivot جديد لكل جزء وهكذا...



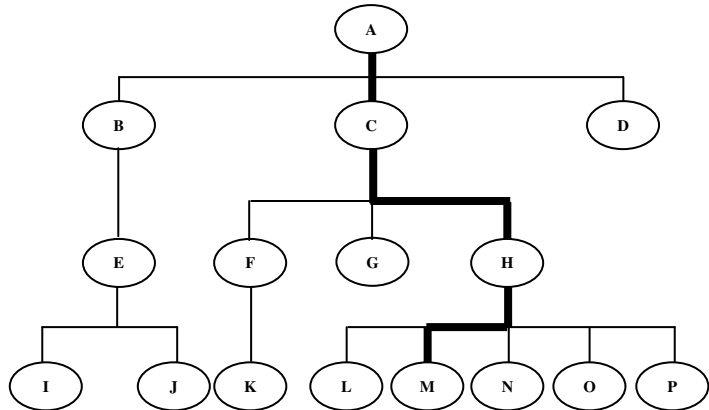
الأشجار Trees



الشجرة هي مكان غير خطي لتخزين المعلومات ،
وتستخدم الأشجار لتمثيل المعلومات التي لها علاقات تشعبية
بحيث أن كل عنصر "معلومة" في الشجرة له أب واحد وقد
يكون له صفر أو أكثر من الأبناء.
الجزر Root هو العنصر الوحيد الذي لا يوجد له أب.

مصطلحات الأشجار

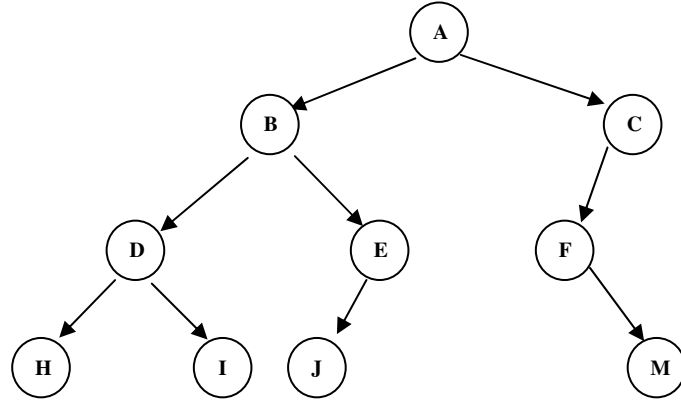
عناصر الأشجار تسمى خلايا **Nodes** ، وكل خلية
لها مسار **Path** واحد فقط يوصلها بالجزر **Root**.
والمسار هو عبارة عن مجموعة خلايا متتابعة
للوصول الى خلية معينة. طول المسار **Path Length** هو
عبارة عن عدد الوصلات من الجزر الى الخلية المراد معرفة
طول مسارها والذي يساوي عدد الخلايا ناقص واحد.
في الشجرة التالية المسار (M,H,C,A) يوصل الخلية M
بالجزر A طوله 3.



عمق الخلية Depth هو طول مسارها الى الجزر ، مثلا الخلية E عمقها 2. الجزر A عمقه 0.
المستوى Level هو كل الخلايا التي لها نفس العمق. المستوى الثاني عبارة عن {E,F,G,H}.
ارتفاع الشجرة Height هو أكبر عمق موجود للشجرة، وفي الشجرة السابقة يساوي 3.
الشجرة التي يوجد بها خلية واحدة فقط ارتفاعها يساوي 0.
والشجرة التي لا تحتوي على أي خلية يعرف ارتفاعها (-1).
درجة الخلية **Degree** هو عدد أبنائها الخلية H درجتها 5.
الورقة **Leaf** هي الخلية التي درجتها صفر أي لا يوجد لها أبناء.

الأشجار الثنائية Binary Trees

الأشجار الثنائية هي أشجار كسابقتها في التعريف إلا أن عدد الأبناء لأي خلية لا يتجاوز الاثنين.



من الشكل السابق نستطيع تعريف تركيبية البيانات كالتالي :

```
struct Node
{
    char data;
    Node* Right;
    Node* Left;
};
```

تطبيقات Binary Trees

1- شجرة Huffman لضغط البيانات

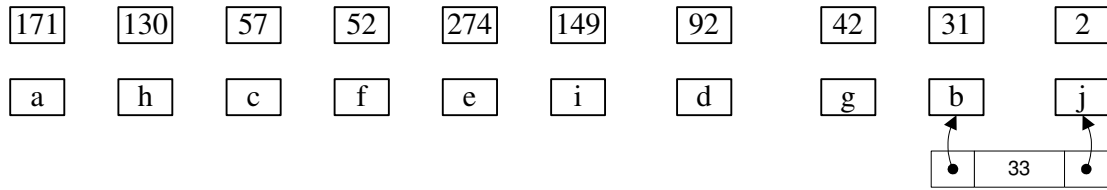
لشرح كيفية عمل طريقة Huffman نفرض انه يوجد لدينا ملف وحجمه 1000 byte يحتوي على الحروف (a,b,c,d,e,f,g,i,j).

1- نقوم بعمل إحصائية عن الملف المراد تقليص حجمه وذلك بعد تكرار كل حرف.

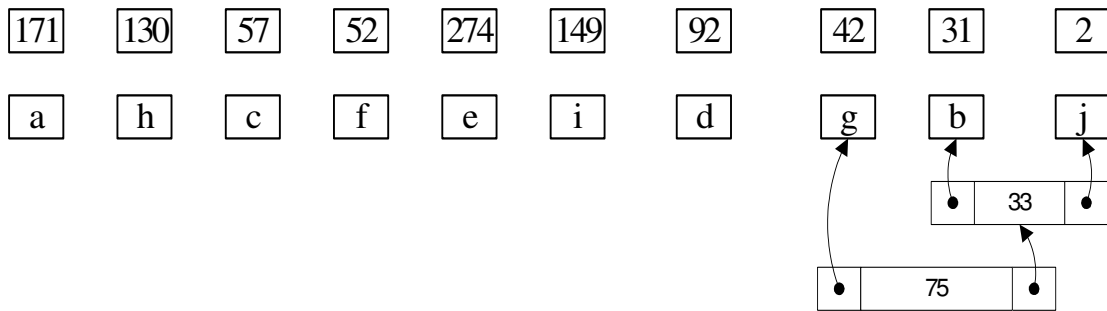
a	b	c	d	e	f	g	h	i	j
171	31	57	92	274	52	42	130	149	2

2- نقوم ببناء Binary Tree وذلك عن طريق إختيار الحروف ذات الأقل تكرار وتوصيلها ببعض.

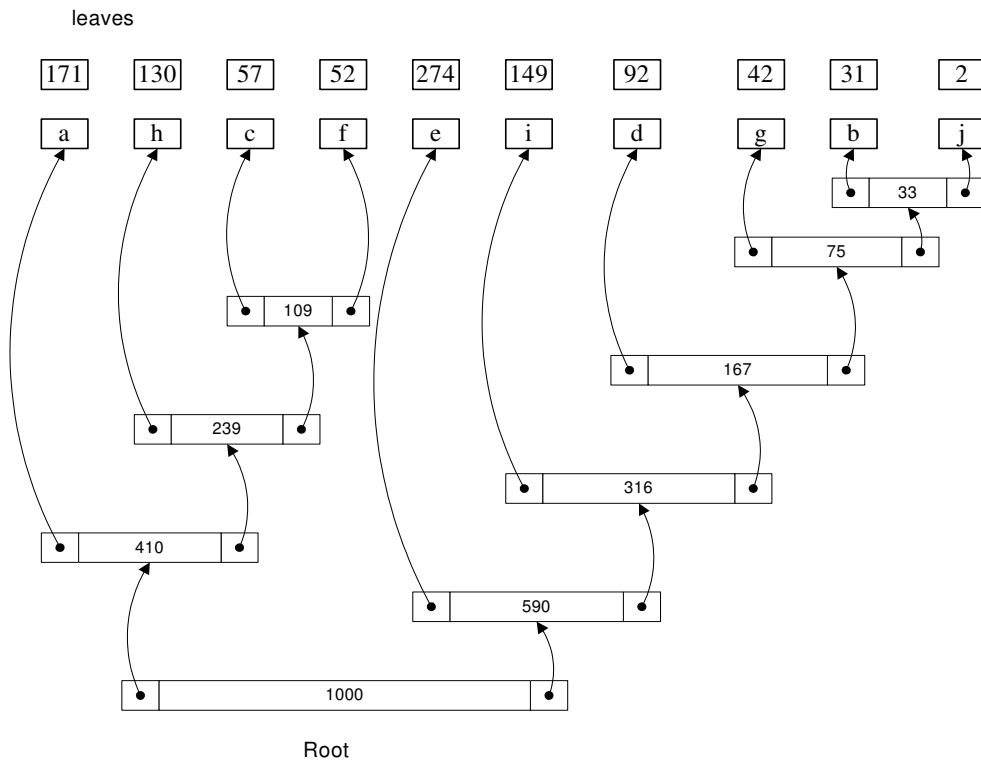
في هذا المثال حرف j وحرف b يمثلان أقل تكرار ، نقوم بتوصيلهما مع وضع مجموع تكرارهما.



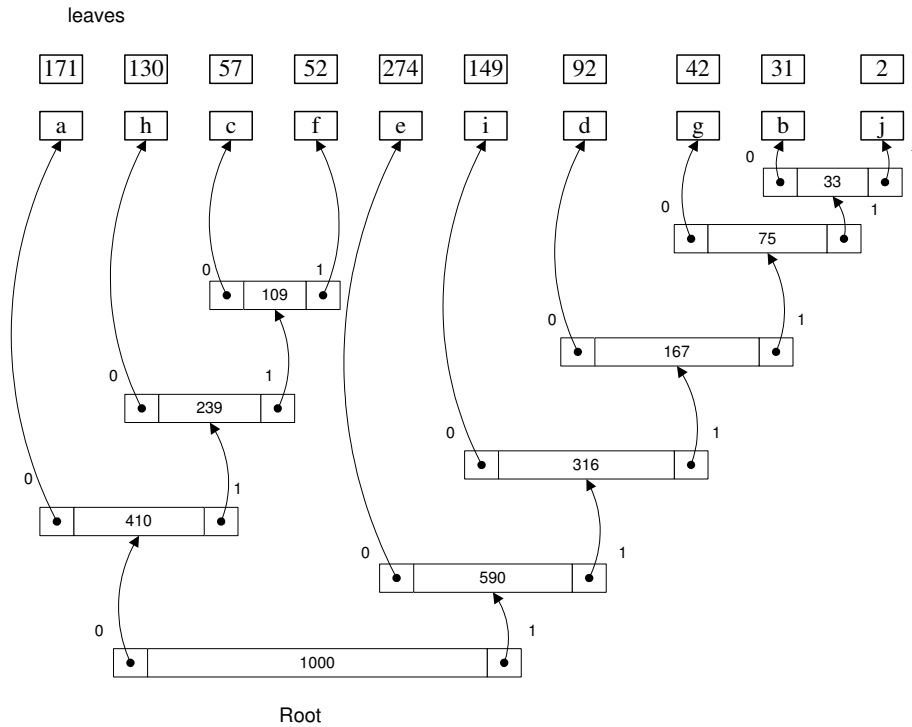
الآن يمثل مجموع حرفي j,b وحرف g الأقل تكرارا نقوم بتوصيلهما بنفس الخطوات السابقة



عند الإنتهاء من توصيل جميع الحروف تكون Binary Tree كالآتي:



3- نقوم بتوزيع 1 على كل فرع من الفروع اليمنى ، و 0 على الفروع اليسرى.



4- نقوم بتسجيل المسارات التي توصل الى كل حرف من الحروف الأصلية للملف ، وذلك بتتبع مسارات Binary Tree من الجذر Root الى الأوراق Leaves.

a	b	c	d	e	f	g	h	i	j
00	111110	0110	1110	10	0111	11110	010	110	111111

عند تكوين الملف المضغوط نستبدل الحروف الأصلية بمساراتها المحسوبة في الخطوة رقم 4.

ونستطيع حساب حجم الملف الجديد ونسبة تقليصه بضرب طول مسار كل حرف في تكراره في الملف

a	b	c	d	e	f	g	h	i	j
00	111110	0110	1110	10	0111	11110	010	110	111111
2	7	4	4	2	4	5	3	3	6
171	31	57	92	274	52	42	130	149	2
342	217	228	368	548	208	210	390	447	12
									2970

أي أن الملف الجديد سيكون حجمه $2970/8 = 372$ بايت ، بمعنى أن الحجم الجديد يعادل 37.2% من الحجم الأصلي.

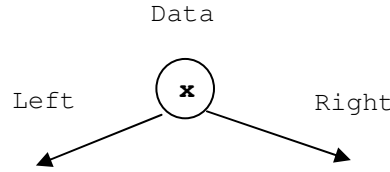
فكرة عمل هذه الطريقة تتلخص في أنها تستغل الحروف الموجودة بكثرة في الملف وتضعها في أقصر مسار في Binary Tree مما يعني أنها (الحروف) سيتم إعطاؤها أقل حيز تخزيني ممكن.

المصدر <http://www.data-compression.com/lossless.html>

إسترجاع المعلومات من الأشجار الثنائية Binary Tree Traversal

والمقصود هنا هو زيارة كل خلية واسترجاع معلوماتها "للطباعة مثلاً" مرة واحد فقط. ولنفرض أن تمثيل الشجرة الثنائية كما في الشكل التالي :

```
struct Node{
    Node* left;
    Node* right;
    char data;
};
```



فمن الشكل السابق يمكننا إسترجاع البيانات من الشجرة الثنائية بست طرق وهي :

Left Right Data
Left Data Right
Right Left Data
Right Data Left
Data Left Right
Data Right Left

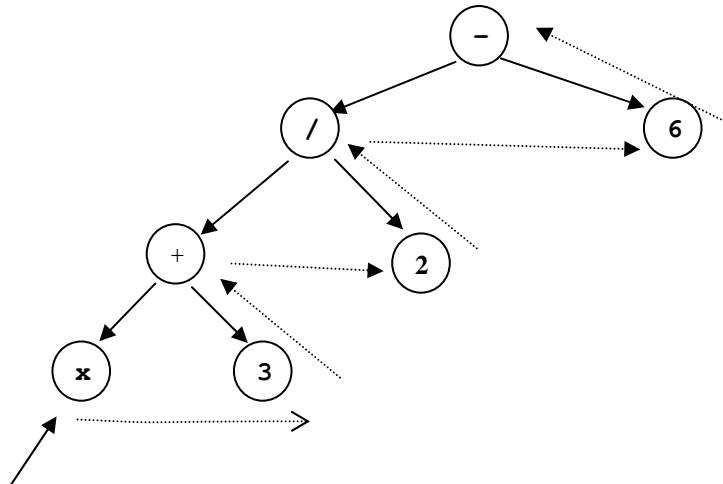
وبإستبعاد الطرق التي توجد بها Right قبل Left يتبقى لنا ثلاث طرق وهي :

الأولى وتسمى Post Order وتمثل Left Right Data
الثانية وتسمى Pre Order وتمثل Data Left Right
والثالثة وتسمى In Order وتمثل Left Data Right
وسميت هذه الطرق نسبة إلى موقع الـ Data ، ففي الطريقة الأولى تقع Data الأخيرة... وهكذا..

طريقة Post Order

```
void PostOrder(Node* ptr)
{
    if(ptr){
        PostOrder(ptr->left);
        PostOrder(ptr->right);
        cout << ptr->data << " ";
    }
}
```

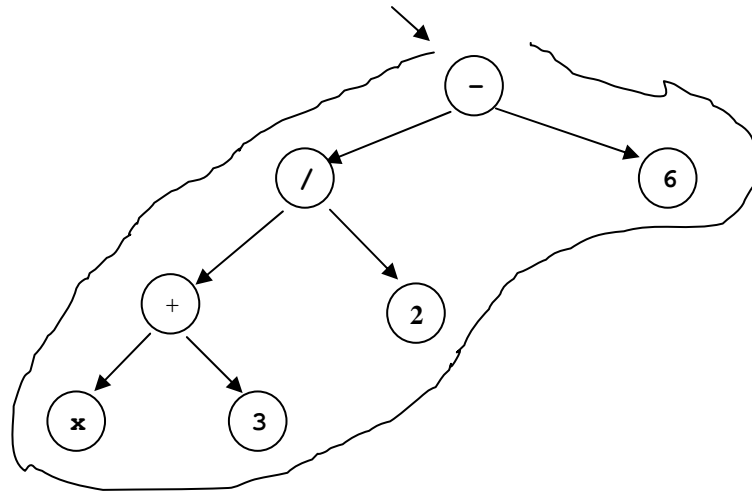
بالنظر إلى عمل هذه الطريقة نجدها تسترجع البيانات من المستويات الأعمق أولاً ، ثم التي تليها وهكذا، مع أولوية الجهة اليسرى للشجرة.



طريقة Pre Order

```
void PreOrder(Node* ptr)
{
    if(ptr){
        cout << ptr->data << " ";
        PreOrder(ptr->left);
        PreOrder(ptr->right);
    }
}
```

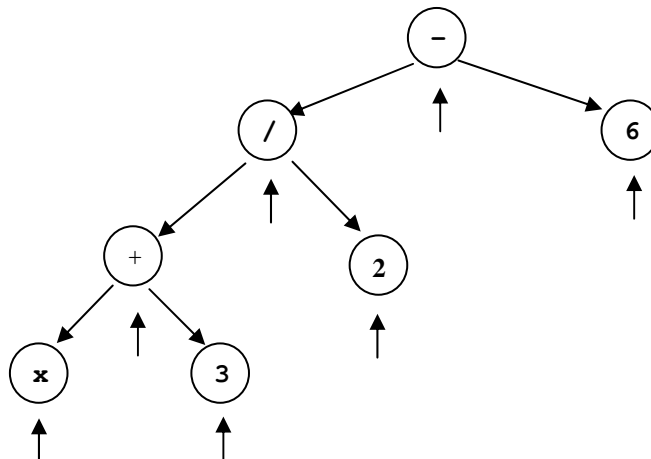
يمكن إسترجاع البيانات بالرسم وذلك برسم محيط حول الشجرة بادية من الجذر ، مع أولوية الجهة اليسرى للشجرة.



In Order

```
void InOrder(Node* ptr)
{
    if(ptr){
        InOrder(ptr->left);
        cout << ptr->data << " ";
        InOrder(ptr->right);
    }
}
```

يمكن إسترجاع البيانات من الرسم وذلك بإسترجاع البيانات الموجودة في أقصى اليسار ، بغض النظر عن المستوى ... وهكذا.



```

#include <iostream.h>
struct Node{
    Node* left;
    Node* right;
    char data;
};

Node* NEW(char c)
{
    Node* p = new Node;
    p->data=c;
    p->left=p->right=NULL;
    return p;
}

void AddLeft (Node* p,char c)
{
    Node* q=NEW(c);
    p->left=q;
}

void AddRight (Node* p,char c)
{
    Node* q=NEW(c);
    p->right=q;
}

void PostOrder(Node* ptr)
{
    if(ptr){
        PostOrder(ptr->left);
        PostOrder(ptr->right);
        cout << ptr->data << " ";
    }
}

void PreOrder(Node* ptr)
{
    if(ptr){
        cout << ptr->data << " ";
        PreOrder(ptr->left);
        PreOrder(ptr->right);
    }
}

void InOrder(Node* ptr)
{
    if(ptr){
        InOrder(ptr->left);
        cout << ptr->data << " ";
        InOrder(ptr->right);
    }
}

void main()
{
    Node* Head = NEW('-'); // 5*x / 2 - 6
    AddLeft (Head, '/');
    AddRight (Head->left, '2');
    AddLeft (Head->left, '+');
    AddLeft (Head->left->left, 'x');
    AddRight (Head->left->left, '3');
    AddRight (Head, '6');

    cout << "Postorder\t";
    PostOrder(Head);
    cout << endl;

    cout << "Preorder \t";
    PreOrder(Head);
    cout << endl;

    cout << "Inorder \t";
    InOrder(Head);
    cout << endl;
}

```

Postorder x3+2/6-
 Preorder -/+x326
 Inorder x+3/2-6

البحث Searching
- Sequential Searches
- Binary Search