POINTERS
&
LISTS

POINTERS

Pointer Arithmetic

1. Addition of a number to a pointer. For example,

```
int i = 4, *j, *k ;
    j = &i ;
    j = j + 1 ;
    j = j + 9 ;
    k = j + 3 ;
```

2. Subtraction of a number from a pointer. For example,

```
int i = 4, *j, *k ;
    j = &i ;
    j = j - 2 ;
    j = j - 5 ;
    k = j - 6 ;
```

Pointer Arithmetic

3. Subtraction of one pointer from another

```
main( )
    {
        int arr[ ] = { 10, 20, 30, 45, 67, 56, 74 } ;
        int *i, *j ;

        i = &arr[1] ;
        j = &arr[5] ;
        Cout<<(j-i)<<(*j-*i)<<endl;
    }
```

Pointer Arithmetic

4. Comparison of two pointer variables

```
main( )
    {
        int arr[ ] = { 10, 20, 36, 72, 45, 36 } ;
        int *j, *k ;

        j = &arr [ 4 ] ;
        k = ( arr + 4 ) ;

        if ( j == k )
           printf ( "The two pointers point to the same location" ) ;
        else
           printf ( "The two pointers do not point to the same location" ) ;
    }
```

Pointer Arithmetic

- *Do not attempt following operations on pointers !*
  - Addition of two pointers
  - Multiplication of a pointer with a constant
  - Division of a pointer with a constant

C++ Array of pointers

```
int *ptr[3];
```

```
void  main ()
{
  int  var[3] = {10, 100, 200};
  int *ptr[3];

  for (int i = 0; i < 3; i++)
  {
    ptr[i] = &var[i];          // assign the address of integer.
  }
  for (int i = 0; i < 3; i++)
  {
    cout << "Value of var[" << i << "] = ";
    cout << *ptr[i] << endl;
  }
}
```

الناتج

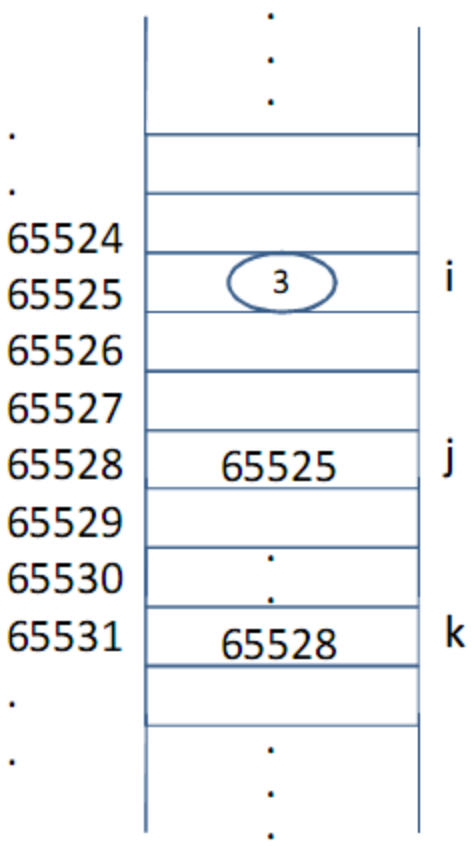
```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

```
#include <iostream>
Void  main ()
{
  char *names[4] = {
                "Zara Ali",
                "Hina Ali",
                "Nuha Ali",
                "Sara Ali",
  };

  for (int i = 0; i < MAX; i++)
  {
     cout << "Value of names[" << i << "] = ";
     cout << names[i] << endl;
  }
}
```

```
Value of names[0] = Zara Ali
Value of names[1] = Hina Ali
Value of names[2] = Nuha Ali
Value of names[3] = Sara Ali
```

Pointer to a Pointer!

```
main( )
{
    int i = 3, *j, **k ;

    j = &i ;
    k = &j ;


    Cout<<“Address of I  = “<<*k<<endl;
    Cout<<“Address of j = “<<&j<<endl;
    Cout<<“Address of j  = “<<k<<endl;
    Cout<<“Address of k = “<<&k<<endl;
    Cout<<“value of j = “<<j<<endl;
    Cout<<“value of k = “<<k<<endl;
    Cout<<“value of i = “<<*(&i)<<endl;
    Cout<<“value of i = “<<*j<<endl;
    cout<<“value of i= “<<**k;
}
```



C++ Pointers to Pointers

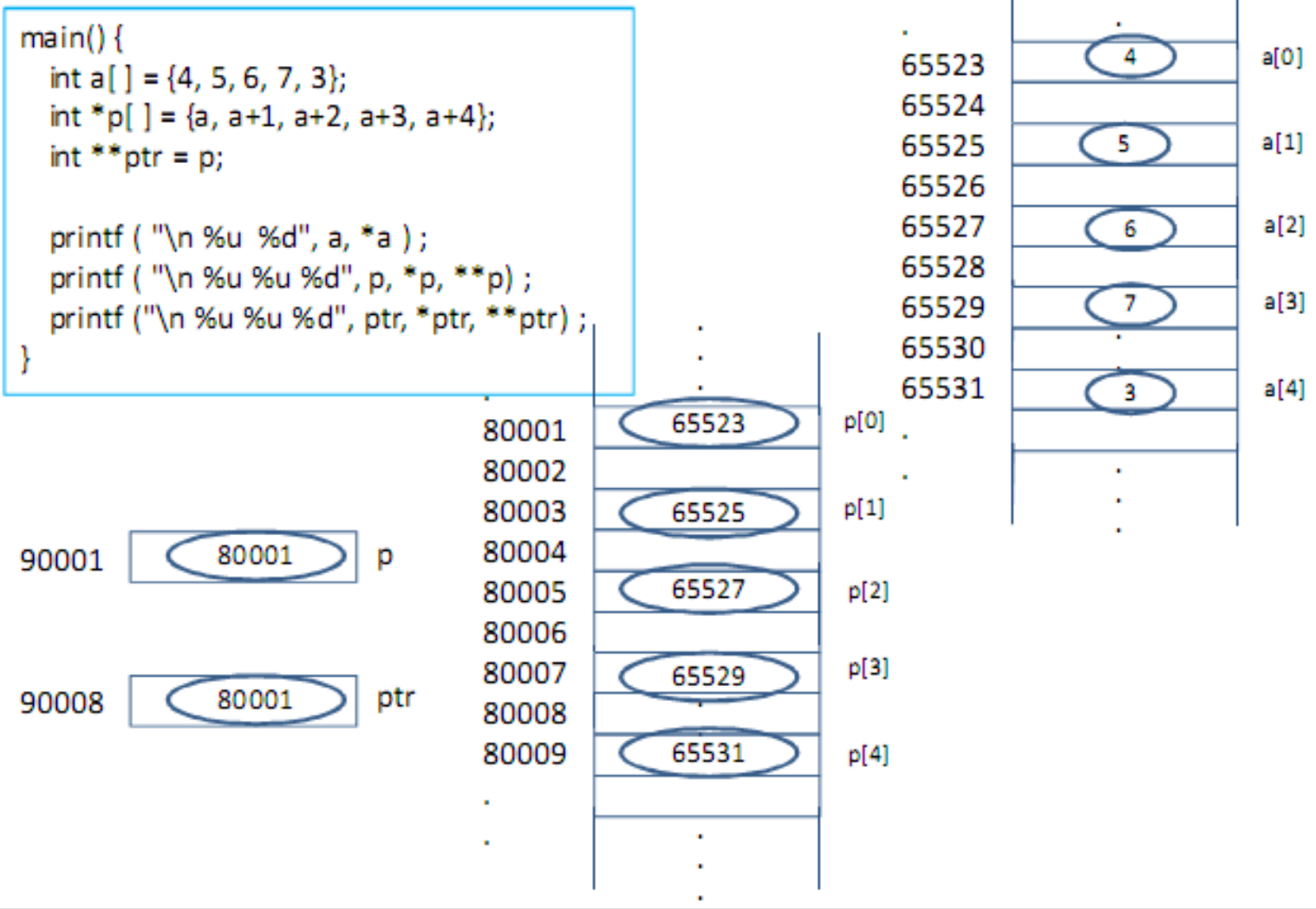
```
int **var;
```

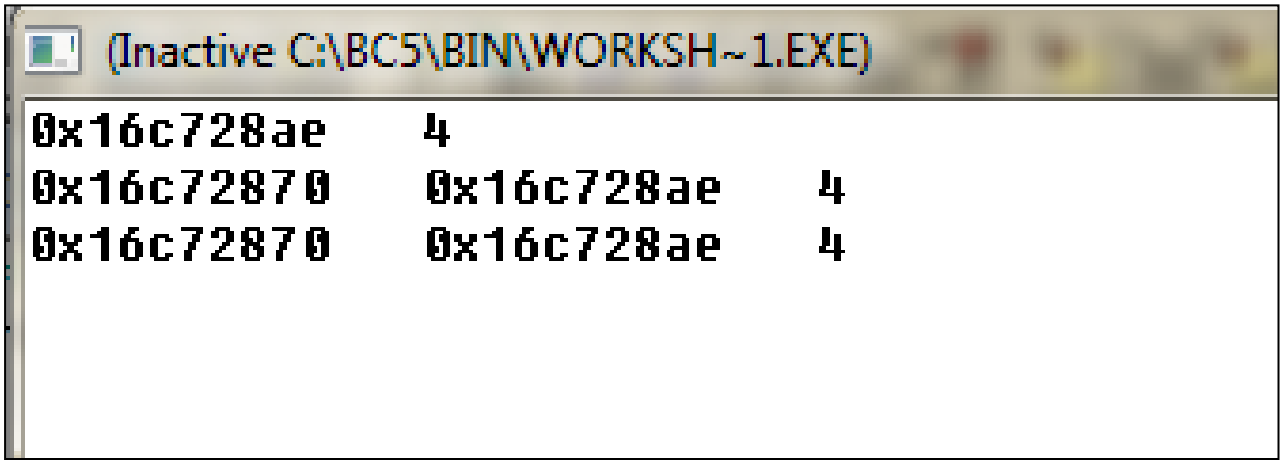
```
#include <iostream>
int main ()
{
  int  var;              int  *ptr;              int  **pptr;
  var = 3000;

  ptr = &var;      // take the address of var

  // take the address of ptr using address of operator &
  pptr = &ptr;

  cout << "Value of var :" << var << endl;
  cout << "Value available at *ptr :" << *ptr << endl;
  cout << "Value available at **pptr :" << **pptr << endl;
}
```

```
Value of var :3000
Value available at *ptr :3000
Value available at **pptr :3000
```

Arrays: Examples

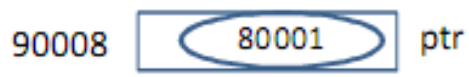
```
main() {
   int a[ ] = {4, 5, 6, 7, 3};
   int *p[ ] = {a, a+1, a+2, a+3, a+4};
   int **ptr = p;

   printf ( "\n %u  %d", a, *a ) ;
   printf ( "\n %u %u %d", p, *p, **p) ;
   printf ("\n %u %u %d", ptr, *ptr, **ptr) ;
}
```

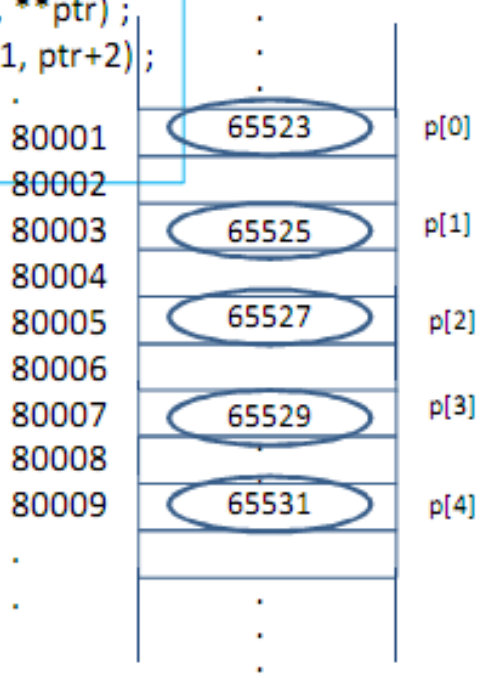


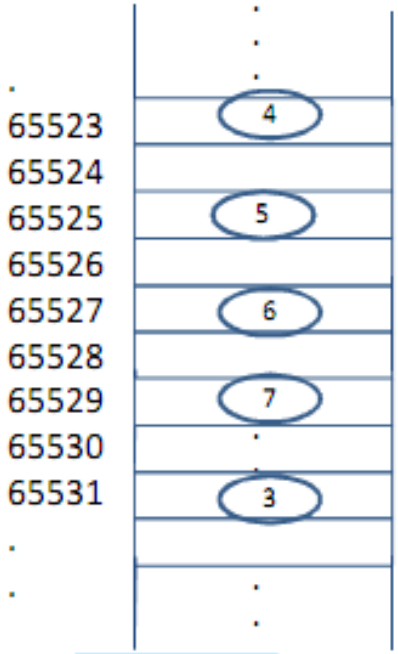

(Inactive C:\BC5\BIN\WORKSH~1.EXE)
0x16c728ae 4
0x16c72870 0x16c728ae 4
0x16c72870 0x16c728ae 4

Arrays: Examples

```
main() {
   int a[ ] = {4, 5, 6, 7, 3};
   int *p[ ] = {a, a+1, a+2, a+3, a+4};
   int **ptr = p;

   printf ( "\n %u  %d", a, *a ) ;
   printf ( "\n %u %u %d", p, *p, **p) ;
   printf ("\n %u %u %d", ptr, *ptr, **ptr) ;
   printf ("\n %u %u %u", ptr, ptr+1, ptr+2) ;

}
```

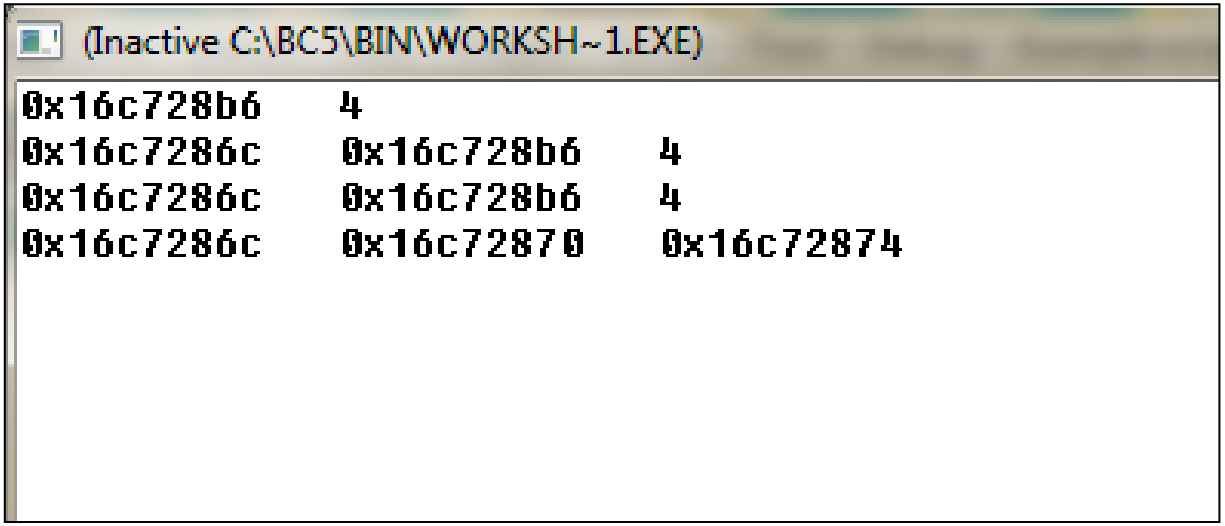







```
ptr++;

*ptr++;

*++ptr;

++*ptr;
```

```
**ptr++;

*++*ptr;

**++ptr;

++**ptr;
```


(Inactive C:\BC5\BIN\WORKSH~1.EXE)
0x16c728b6 4
0x16c7286c 0x16c728b6 4
0x16c7286c 0x16c728b6 4
0x16c7286c 0x16c72870 0x16c72874

LISTS

القائمة الخطية: Linear list

هي مجموعة من العناصر البيانية (items ,nodes, elements) المتسلسلة و المرتبة تربط عناصرها علاقة تجاور بحيث يسبق كل عنصر عنصرا اخر عدا العنصر الأول الذي لا يسبقه عنصر و العنصر الأخير الذي لايليه عنصر فلو مثلنا كل عنصر على شكل عقدة (node) فان القائمة تصبح مجموعة من العقد(n).

x[1].x[2].x[3]……..x[k-1].x[k].x[k+1]……..x[n]

فالعقدة الاولى هي x[1] و العقدة الأخيرة هي x[n] اما العقدةx[k] عندما 1<=k<=n فان العقدة التي تسبقها هي x[k-1] والتي تليها هي x[k+1] .

ان كل مجموعة من البيانات و المعلومات يمكن تسميتها قائمة (list)فمثلا:

@ مجموعة أسماء طلبة كلية ما مرتبة حسب الحروف الابجدية.

@ مجموعة أسماء المشتركين في دليل الهاتف مرتبة وفق نسق معين.

انواع القوائم الخطية : Type of linear

أ-القوائم غير الموصولة Non-Linked-List

ب-القوائم الموصولة Linked List

القوائم غير الموصولة Non-Linked-List

وهي القوائم التي لا تستخدم المؤشرات و تكون على شكل بيانات متتابعة و متجاورة (sequential) و تستخدم المصفوفات في تمثيلها.كما يستخدم هذا النوع عند معالجة البيانات التي لا تتعرض للتغيير كثيرا لصعوبة عمليات الحذف و الإضافة إذ قد تكون المواقع التالية في ذاكرة الحاسوب مشغولة أصلا مما يتعذر استخدامها لأغراض الحذف و الإضافة.

القوائم الموصولة Linked List

وهي القوائم التي تستخدم المؤشرات (pointers)لتسهيل عمليات الإضافة والحذف والتعديل إذ يكون لكل عنصر مؤشر يحدد موقع العنصر التالي ،ووجود المؤشرات يلغي الحاجة لخزن بيانات القائمة في مواقع خزنية متجاورة.

العمليات التي يمكن إجراؤها على القوائم الخطية:

1- البحث search : هي عملية بحث داخل الهيكل البياني بقصد الوصول إلى عنصر (عقدة) معين فيه بموجب قيمة أحد الحقول يسمى حقل المفتاح(key field)أي أن البحث يتم وفق المحتويات و ليس العنوان.

2-إدخال (إضافة) Addition : لإضافة عنصر(عقدة)جديد إلى الهيكل البياني مثل تسجيل طالب جديد في المدرسة.

3-حذف Deletion: حذف عنصر(عقدة) من الهيكل البياني ،مثل نقل طالب إلى مدرسة اخرى.

4-دمج Merge: دمج بيانات هيكلين او اكثر لتكوين هيكل بياني واحد.

5-فصل Split: تجزئة بيانات هيكل بياني إلى هيكلين او اكثر.

6-إحتساب Counting: احتساب عدد العناصر او العقد في الهيكل البياني.

7-نسخ Copying: نسخ بيانات الهيكل البياني الى هيكل بياني اخر.

8-ترتيب Sort: ترتيب عناصر (عقد)الهيكل البياني وفق قيمة حقل (field)او مجموعة حقول.

9-الوصول Access: تتطلب أحيانا الحاجة للوصول إلى عنصر(عقدة)بياني في الهيكل البياني لعدة أغراض لاختباره مثلا أو تغييره...الخ.

MEMORY ALLOCATIONS ادارة الذاكرة

ادارة الذاكرة في C++

- عندما يكون البرنامج في مرحلة التشغيل فانه يحجز اماكن في الذاكرة لكل امر و لكل بيان ، و عندما ينتهي تشغيل البرنامج تتحرر الذاكرة من تلك البيانات و نفقدها بشكل نهائي ، تخصيص اماكن في الذاكرة بهذه الطريقة يسمي بالتخصيص(التوزيع) الغير حر(الثابت) للذاكرة **static allocation**.
- تخصيص اماكن بالذاكرة حسب حاجة المبرمج يطلق علية التوزيع الحر/الديناميكي **dynamic allocation**.

التوزيع الحر(المتغير) dynamic

- توفر C دوال مبنية داخليا (في المكتبة malloc.h) لتخصيص اماكن بالذاكرة بشكل حر (الدالة malloc) و كذلك تحرير الذاكرة (الدالة free).
- توفر C++ دوال مبنية داخليا لتخصيص اماكن بالذاكرة بشكل حر (الدالة new) و كذلك تحرير الذاكرة (الدالة delete).

التخصيص الخزني التسلسلي sequential allocation of storage

ان ابسط الطرق لخزن القائمة الخطية هو استخدام الخزن التسلسلي في ذاكرة الحاسوب اي يتم الخزن في مواقع متتابعة (متسلسلة) ويمكن ان نعرف موقع اي عنصر اذا عرفنا موقع العنصر الاول الذي هو عنوان البداية base address ومواقع العناصر التالية ستكون منسوبة له. فالعنصر k سيكون موقعه تاليا لموقع العنصر k-1 وهكذا بقية العناصر.

المزايا advantages:

١- اكثر سهولة في التمثيل والتطبيق .
٢- يكون اقتصاديا اكثر لانه يستخدم مساحة خزنية اقل.
٣- اكثر كفاءة في الوصول العشوائي.
٤- مناسب جدا عند التعامل مع المكدس.

المساوئ:disadvantage:

١- صعوبة تنفيذ عمليات الاضافة والحذف.
٢- يتطلب التعريف المسبق وتحديد عدد العناصر المطلوب خزنها.

التخصيص الخزني الديناميكي:dynamic allocation of storage:

ان الطريقة الاخرى للخزن هي استخدام رابط link او مؤشر مع كل عنصر يحتوي عنوان موقع العنصر التالي لذلك لا توجد ضرورة لخزن البيانات في مواقع متعاقبة (متسلسلة) بل يمكن خزن اي عنصر بياني في اي موقع ، ولهذا فكل عنصر (عقدة) يتكون من جزاين هما:

١- الجزء الاول : يحتوي البيانات data

٢- الجزء الثاني : حقل يحتوي على عنوان موقع العنصر التالي link

المزايا : سهولة تنفيذ عمليات الاضافة والحذف لأي عنصر اذ لايتطلب اكثر من تحديث قيمة حقل المؤشر الذي يعطي عنوان الموقع التالي.

المساوئ: يحتاج الى مساحة خزنية اكبر لتمثيل حقل المؤشر اضافة الى البيانات الاساسية.

المقارنة بين الخزن التسلسلي والخزن الديناميكي

أ- المساحة الخزنية Amount of storage:

ب-عمليات الاضافة والحذف:

ج- الوصول العشوائي للعناصر Random Access:

د- الدمج والفصل MERG AND SPLIT

مثال:

```
double *ptr;

ptr=(double*) malloc(sizeof(double));
```

اختيارية

تحرير وتوزيع الذاكرة في C

مثال:

```
struct emp
{
    int data;
    char name[20];
};
emp *t;                          //create a pointer to the struct emp
t = (emp*)malloc(sizeof(emp));       //allocate space for struct emp
t->data=50;
strcpy(t->name,"hanaa");

…

free(t);     // تحرير الذاكرة
```

مثال:

تحرير وتوزيع الذاكرة في C++

```
struct emp
{
    int data;
    char name[20];
};
emp *t;                    //create a pointer to the struct emp
t = new emp;               //allocate space for struct emp
t->data=50;
strcpy(t->name,"hanaa");

…

delete  t ;     // تحرير الذاكرة
```

- ✓ عند تمثيل المكدس stack و الطابور queue بواسطة المصفوفات فان حجمها يكون محدد و يحجز **statically** اثناء الترجمة فاذا كان الحجم المحدد للمصفوفة كبير و لم نحتاجه بالكامل فهذا يسبب ضياع في الذاكرة **memory wastage** و اذا كان حجم المصفوفة صغير و احتجنا الي حجم اكبر فان هذا يسبب مشكلة **insufficient memory**.

- ✓ عملية الاضافة والحذف من المصفوفة اما تسبب increment or shift لعناصر المصفوفة.

- ✓ عند استخدام ال linked list فإنها تحجز **dynamically** حسب الحاجة اثناء ال run فنجد انه لا وجود للمشاكل السابقة **memory wastage or insufficient memory**. و كذلك عمليتي الاضافة و الحذف تتم بدون أي مشكلة.

هياكل البيانات

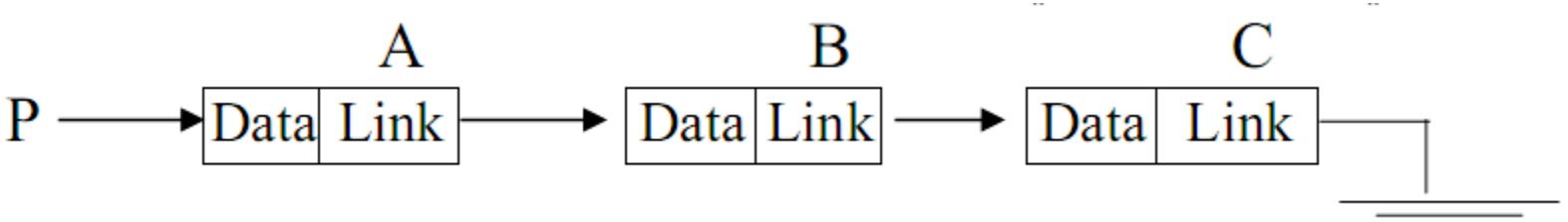
- هياكل البيانات (الكائنات) مثل القوائم ، المجموعات ، .. تعتبر نوع بيان مجرد abstract data type مثل ... , float , int أي نستطيع اشتقاق انواع منها.

- منها نوعان :
  1. خطي أي نخزن البيانات و نعبرها بطريقة خطية (تسلسل) مثل ... , stack , linked list , array.

  2. غير خطية أي نخزن البيانات و نعبرها بطريقة غير خطية مثل tree , graph ,....

الهياكل الموصولة linked structures

ان لغة C++ توفر اسلوب استخدام المؤشر pointer للتعامل مع الهياكل البيانية الموصولة . فالمؤشر هو عبارة عن نوع من البيانات (data type) قيمته تمثل عنوان موقع عنصر عقدة في الذاكرة.

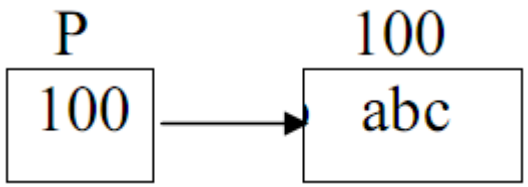



```
struct node{
        int data;
        struct  node*link;
};
```

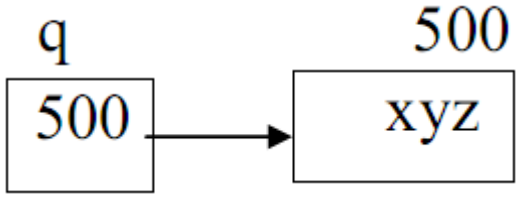
مثال:

```
int *p, *q;
```

ويتضمن تعريف مؤشرين p,q إذ يعني كل منهما موقع محتوياته هي قيمة عددية تمثل عنوان موقع في الذاكرة (memory address) ولنفرض انهما يشيران الى الموقعين التاليين:



حيث الموقع p يحتوي القيمة 100 التي هي عنوان موقع في الذاكرة محتوياته abc.

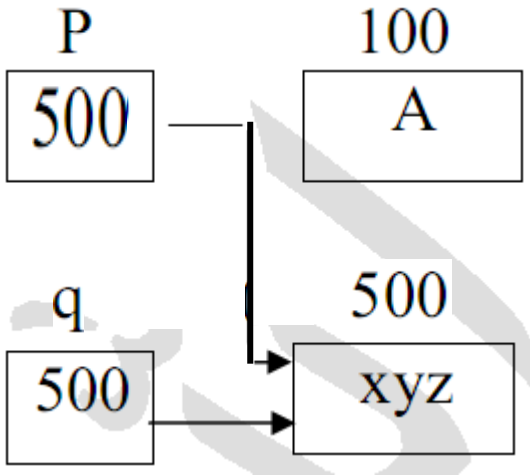


اما الموقع q فيحتوي القيمة 500 التي هي عنوان موقع في الذاكرة محتوياتهxyz.

p=q;

تعني ان قيمة p ستأخذ قيمة q

اي ان المؤشرين p,q سيشيران الى نفس الموقع 500



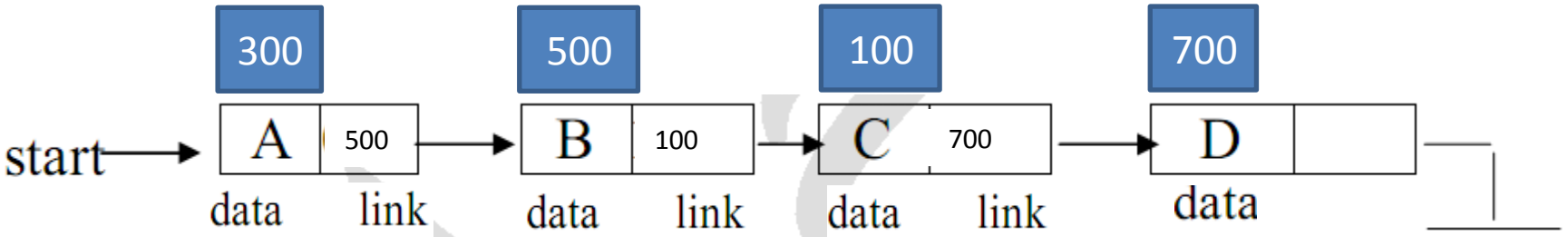
القائمة الموصولة linked list

هي مجموعة من العناصر (العقد) التي كل منها يحتوي البيانات data والمؤشر link الذي يشير الى العنصر (العقدة) التالي في القائمة.
فالعنصر x يتكون من الجزأين link,data

X:

Data(x)	Link(x)

مثال: لنأخذ قائمة موصولة



محتويات حقل المؤشر للعنصر الثاني هو(100) ويدل على موقع العنصر الثالث:(link(B)=100

محتويات حقل المؤشر للعنصر الرابع هو(NULL)اي لا يوجد عنصر بعد العنصر الرابع

القوائم الخطية الموصولة Linear linked list

- القائمة الخطية هي نوع من انواع هياكل البيانات الخطية تستخدم لتنظيم البيانات في الذاكرة و كذلك في تمثيل / محاكاة انواع اخري من هياكل البيانات .

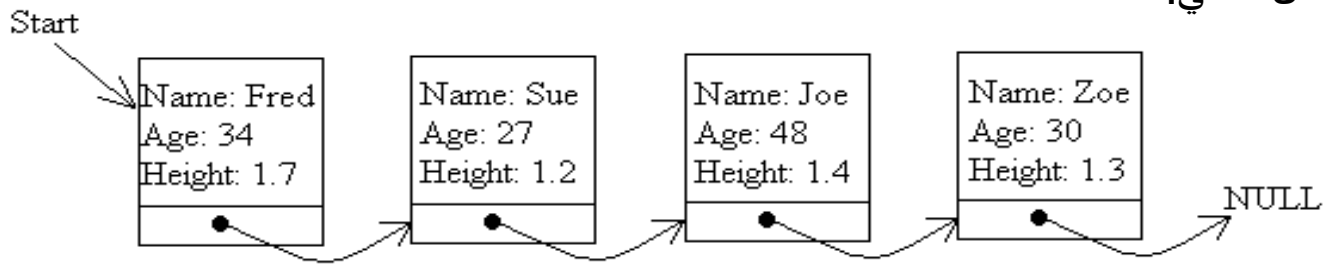
- ليس شرط ان تكون عناصر القائمة متجاورة في الذاكرة .

- تتكون من عدد من العناصر تسمي بالعقد **nodes** مرتبطة مع بعضها عن طريق مؤشرات.

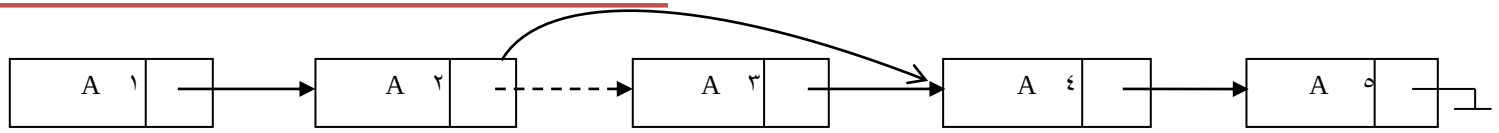
- العقدة (**node**) عبارة عن تركيبة struct أو كائن من طبقة تتكون من حقلين او اكثر من البيانات تسمي بـ 'data /Info' field و حقل آخر للمؤشر يعرف بـ 'Link/address' field .

- الحقل الاول يحمل قيم البيانات و الحقل الثاني يحمل عنوان العقدة التالية أو القيمة **NULL** اذا كانت القائمة فارغة أو نهاية القائمة.

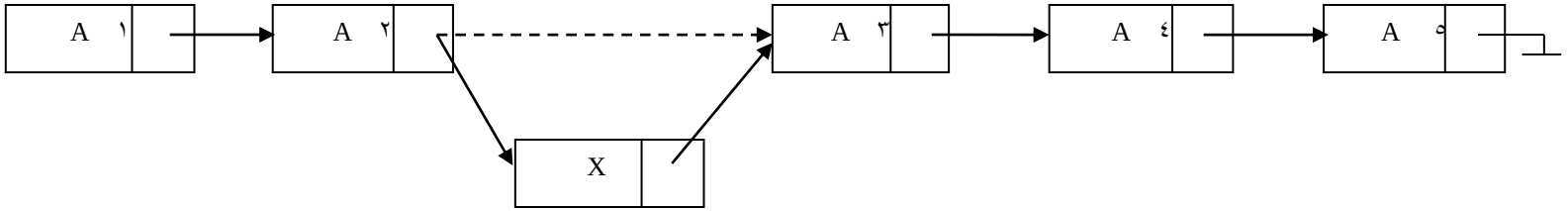
- تأخذ الشكل التالي:



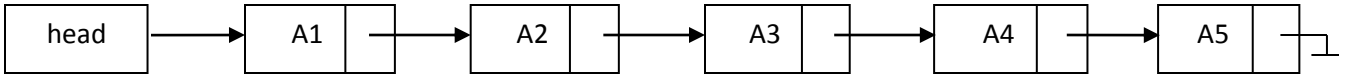
Deletion from a linked list



Insertion to a linked list



Linked List with a header



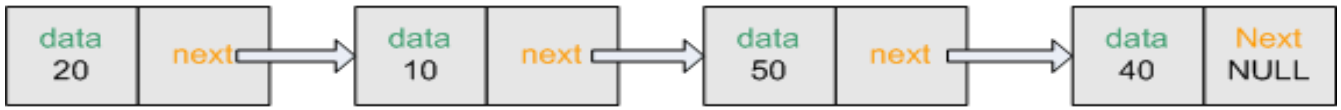
حيث head هو مؤشر للعقدة الأولي في القائمة.

مثال:

الاعلان عن عقدة (حلقة node) تحتوي علي البيان data + مؤشر للعقدة التالية

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

The items of the list عناصر القائمة



Linked list

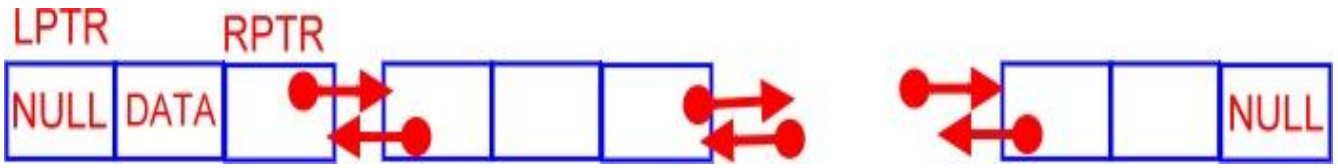
الاعلان عن عقدة Node تحتوي علي بيانات شخص (الاسم ، العمر ، الوزن)
+ مؤشر للعقدة التالية

```
struct Node
{
     char name[20];
     int age;
     float height;
     Node *next;              // Pointer to next node
};
Node *start_ptr = NULL;       // Start Pointer (root)
```

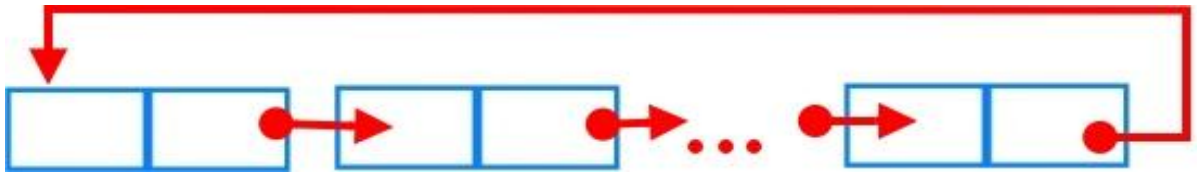
Linked List Type's انواع القوائم الموصولة

- Singly-Linked Lists
- Doubly-Linked Lists or Two way Linked List
- Circularly-Linked Lists
- Circularly-Doubly Linked Lists

قائمة خطية مزدوجة : Doubly-Linked List



قائمة خطية دائرية Circularly-Linked List



اضافة عقدة لبداية القائمة (Insert from front) :

Linked list

```
struct node
{
    int data;
    node *next;
};
typedef struct node *PtrToNode;
typedef PtrToNode List;
```

```
node* inserttofront(List l, int info) //where l is a head
{
   node *temp;
   if (l==NULL)
   { l=new node; l->data=info;  l->next=NULL; }
   else
   {
      temp = new node;
      temp->data = info;
      temp->next=l;
      l = temp;
   }
   return l;    // return with head node
}
```

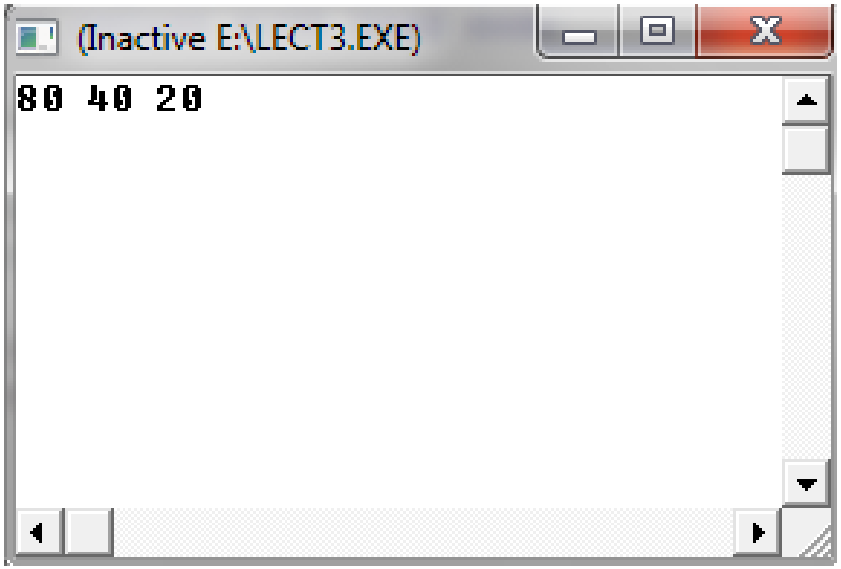
```
void main()
{
   List Head;
   Head=NULL;              // LIST IS EMPTY
   Head=inserttofront(Head,20);
   Head=inserttofront(Head,40);
   Head=inserttofront(Head,80);
}
```

Traverse the list عبور القائمة

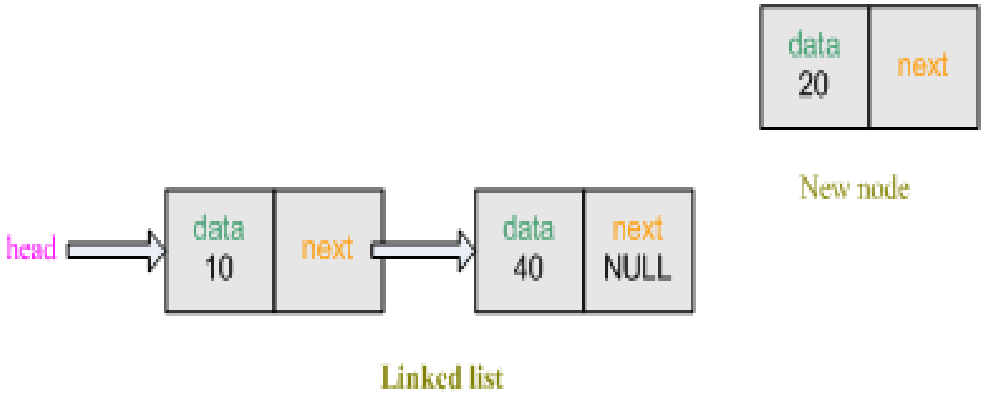
Linked list

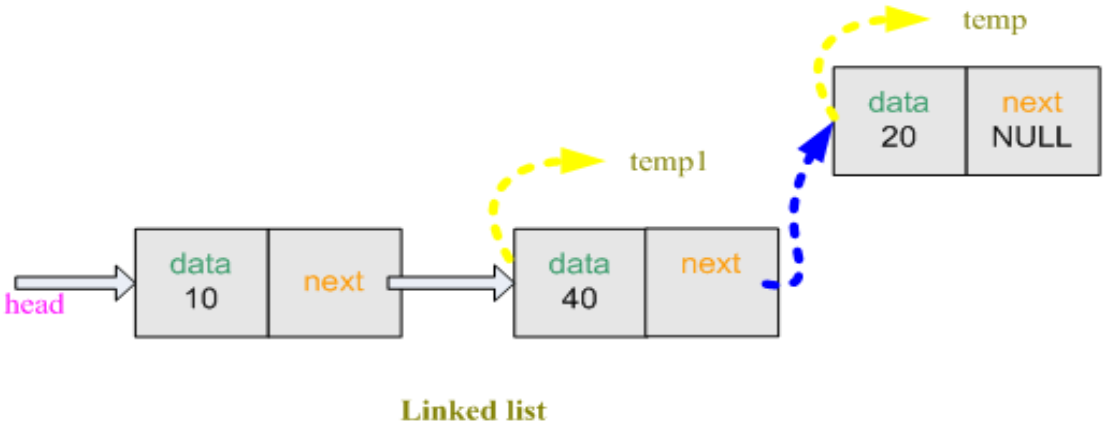
```
void traverse(List l )
{
   node *temp=l;
   while( temp!=NULL )
   {
      cout<< temp->data<<" ";
      temp = temp->next;
   }
}
```

```
…
void main()
{
    List Head;
    Head=NULL;        // LIST IS EMPTY
    Head=inserttofront(Head,20);
    Head=inserttofront(Head,40);
    Head=inserttofront(Head,80);
    traverse(Head);
}
```



اضافة عقدة جديدة في نهاية القائمة Insert to back





```
node* insert_to_back(List head,int info)
{
        node *temp,*temp2;
        temp = new node;
        temp->data = info;
        temp->next = NULL;
        if(head==NULL)          // empty list
             head=temp;
        else
        {
             temp2 = head;
             while(temp2->next!=NULL)
                  temp2 = temp2->next;
             temp2->next = temp;
        }
        return head;
}
```

```
void main()
{
        List Head;
        Head=NULL;      // LIST IS EMPTY
        Head=inserttofront(Head,20);
        Head=inserttofront(Head,40);
        Head=inserttofront(Head,80);
        traverse(Head);
        cout<<endl;
        Head=insert_to_back(Head,120);
        traverse(Head);
}
```