كلية الهندسة – قسم المعلوماتية

مقرر برمجة 2

ا. د. علي سليمان

محاضرات الأسبوع الأول

الفصل الثاني 2023-2024

محتوى المقرر

1. Course overview.	1. فكرة عامة عن المقرر.
2. Pointers and Strings.	2. المؤشرات والسلاسل المحرفية.
3. Classes and Objects.	3. الأصناف والأغراض
4. Arrays, Pointers, References, And the Dynamic Allocation Operators.	4. المصفوفات، المؤشرات، المراجع، ومعاملات التخصيص الديناميكي
5. Function Overloading ,Copy constructors , and Default Arguments.	5. التحميل الزائد للتوابع، التوابع البانية الناسخة، والوسطاء الافتراضية
6. Operator Overloading.	6. التحميل الزائد للمعاملات
7. Inheritance.	7. الوراثة
8. Virtual Functions and Polymorphism.	8. التوابع الظاهرية وتعدد الأشكال
9. Templates.	9. القوالب
10. Exception Handling.	10. معالجة الاستثناءات
11. The C++ I/O System Basics.	11. أساسيات نظام الدخل / الخرج في لغة C++
12. C++ File I/O.	12. الإدخال والإخراج من الملفات في C++

المحاضرة من المراجع :

[1]- Deitel & Deitel, C++ How to Program, Pearson; 10th Edition (February 29, 2016)

[2]- د.علي سليمان, البرمجة غرضية التوجه في لغة C++ 2009-2010

مقدمة إلى OOP 1

- مفهوم البرمجة غرضية التوجه.
- مفهوم التغليف، تعدد الأشكال، الوارثة.
- استخدام المفاهيم الأساسية للبرمجة بلغة C++، كالتعامل مع الدخل/الخرج، التصريح عن المتحولات المحلية، التعامل مع أنماط البيانات.

مقدمة إلى OOP 2

- التغليف Encapsulation:

التغليف هو الآلية التي تربط الشيفرة والبيانات التي تتعامل معها، وتحميها من كل من التدخل الخارجي و سوء الاستخدام. في اللغة غرضية التوجه يمكن تجميع الشيفرة والبيانات بحيث يتم إنشاء "صندوق أسود" قائم بذاته، عندما يتم ربط البيانات والشيفرة بهذه الطريقة يتم إنشاء غرض object، بمعنى آخر الغرض هو الجهاز الذي يدعم التغليف.

يمكن ضمن غرض ما أن تكون البيانات، الشيفرة أو كلاهما، خاصة private بذلك الغرض أو عامة public. الشيفرة أو البيانات الخاصة تكون مرئية، ويمكن الوصول إليها فقط من قبل الأجزاء الأخرى للغرض، أي أن الشيفرة أو البيانات الخاصة لا يمكن الوصول إليها من قبل أي أجزاء من البرنامج توجد خارج الغرض، أما عندما تكون الشيفرة أو البيانات عامة، فإن الأجزاء الأخرى من البرنامج يمكن أن تصل إليها حتى ولو كانت معرفة ضمن غرض، نموذجياً الأجزاء العامة للغرض تستخدم لتحقيق واجهة متحكم بها للأجزاء الخاصة للغرض.

مقدمة إلى OOP 3

- تعدد الأشكال Polymorphism:

"واجهة واحدة، طرائق متعددة one interface , multiple methods " كمثال من العالم الحقيقي عن تعدد الأشكال هو منظم الحرارة thermostat. فبغض النظر عن طريقة توليد الحرارة في منزلك (غاز، نفط، كهرباء ... إلخ) فإن منظم الحرارة يعمل بالطريقة ذاتها، في هذه الحالة، منظم الحرارة (الذي يمثل الواجهة interface) هو نفسه بغض النظر عن نوع مصدر الحرارة (الذي يمثل الطريقة method) الذي تقوم باستخدامه. أي إذا أردت درجة حرارة المشع 70 درجة مئوية، فأنت تقم بضبط منظم الحرارة على 70 درجة ولا يكون مهماً ما هو المصدر الذي يحقق هذه الحرارة.

كمثال آخر لنفرض لديك برنامج يعرف ثلاثة أنماط مختلفة من المكدسات، أحدها للقيم الصحيحة، الآخر للقيم المحرفية والثالث لقيم الفاصلة العائمة، بإمكانك بفضل تعدد الأشكال تعريف مجموعة من الطرائق أسماؤها ()push، ()pop التي يمكن استخدامها من أجل المكدسات الثلاث جميعاً، تستطيع في برنامجك إنشاء ثلاث نسخ مختلفة من هذه التوابع، واحدة لكل نوع من المكدسات لكن أسماء التوابع ستكون نفسها، وسيقوم المترجم تلقائياً باختيار التابع الملائم بحسب البيانات المخزنة، وبالتالي فإن الواجهة إلى المكدس – التوابع ()push و ()pop– هي نفسها بغض النظر عن نوع المكدس المستخدم.

تساعد تعدد الأشكال على تقليل التعقيد من خلال السماح لنفس الواجهة لأن تصل إلى صنف عام من الأفعال، وتكون مهمة المترجم هي اختيار الفعل المحدد (الطريقة) التي سيقوم باستخدامها في كل حالة ولا يحتاج المبرمج لاختيارها يدوياً، فقط يحتاج لأن يعرف، ويستخدم الواجهة العامة general interface.

مقدمة إلى OOP 4

- الوراثة inheritance:

" الوراثة هي العملية التي يستطيع من خلالها غرض ما الحصول على خصائص غرض آخر، إن هذه العملية مهمة جداً لأنها تدعم مبدأ التصنيف classification والذي يعني أن أغلب المعلومات قابلة للإدارة من خلال تصنيف شجري hierarchical classification.

التفاحة الحمراء هي جزء من التصنيف تفاح apple، الذي هو بدوره جزء من الصنف فاكهة fruit، وهو بدوره أيضاً جزء من تصنيف أكبر هو الطعام food. دون القيام باستخدام التصنيف، فإن كل غرض يجب تعريف جميع خصائصه بشكل صريح، في حين أنه باستخدام التصنيف، فإن كل غرض يحتاج لأن يتم تعريف الخصائص التي تميزه فقط ضمن الصنف. إن مبدأ الوراثة هو ما يجعل من الممكن لغرض أن يكون حالة خاصة من حالة أكثر عمومية، سترى لاحقاً أن الوراثة هي إحدى أهم مفاهيم البرمجة غرضية التوجه.

مقدمة إلى OOP 5

- بعض ميزات البرمجة كائنية التوجه مقارنه مع البرمجة الإجرائية بالنقاط التالية:
  - إعتمادها على الكائنات وليس على الأفعال.
  - تعتبر البرمجة غرضية التوجه هى الأسرع والاسهل في التنفيذ.
  - تؤمن البرمجة غرضية التوجه بنية أكثر وضوحاً للبرامج.
  - تجعل البرنامج خال من التكرارات واسهل في الصيانة والتعديل.
  - تتيح إمكانية خلق تطبيقات مختلفة من نفس البرمجة بتعليمات اقل ووقت اقصر للتطوير والتعديل.

المؤشرات والسلاسل المحرفية

عند الانتهاء من دراسة هذا الفصل سيكون الطالب قادراً على ما يلي :

- تعريف المؤشر والتصريح عنه .
- تمييز العلاقة بين المؤشرات والمصفوفات .
- استخدام المؤشرات كبارمترات للتوابع أو كقيم معادة منها .
- استخدام توابع التخصيص الديناميكي.
- تعريف واستخدام مصفوفات المؤشرات.
- التصريح عن السلاسل المحرفية وإجراء عمليات الإدخال والإخراج.
- استخدام بعض توابع المكتبة القياسية string.h.

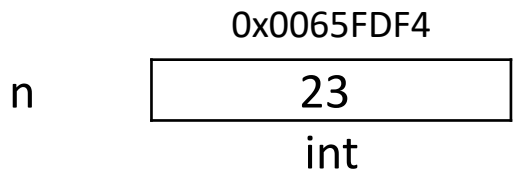
2-1-المؤشرات 1

Pointers

- يعود استخدامها لثلاثة أسباب:
  - ✓ المؤشرات تعطي إمكان قيام التوابع بتعديل وسطاء الاستدعاء.
  - ✓ المؤشرات تدعم التخصيص الديناميكي dynamic allocation.
  - ✓ المؤشرات يمكن أن تحسن من فعالية بعض الإجرائيات.

- الإعلان عن المتغير ينطوي على ثلاثة مفاهيم أساسية:
  - ✓ اسم المتغير.
  - ✓ نوع المتغير.
  - ✓ عنوان المتغير في الذاكرة.

فمثلاً ، التصريح التالي عن المتحول n: `int n=23;` يربط بين الاسم n والنوع int وعنوان المتغير في الذاكرة، وبالتالي يمكن تصور المتغير المعلن عنه كما يلي:



يمثل 1- الصندوق مكان تخزين المتغير في الذاكرة. 2- اسم المتغير على اليسار. 3- عنوان المتغير من الأعلى.
4- نوع المتغير أسفل الصندوق. 5- والقيمة المخزنة تساوي 23 وهي محتوى الصندوق.

Pointers

2-1-المؤشرات 2

- عنوان المتغير يمكن التعامل معه بعامل العنوان & (address operator) .
  - ✓ طباعة عنوان المتغير n بالأمر التالي: `cout << & n ;` .
  - ✓ عامل العنوان & يسبق اسم المتغير لينتج العنوان وله أسبقية على عامل النفي المنطقي وعامل الزيادة المسبقة ++.
- مثال للاستخدام معمل العنونة:

```
#include "stdafx.h"
#include<iostream>
using namespace std;
int main() {   int n=23;
        cout<<"the value of n is: n="<<n<<endl;
        cout<<"the address of n is: &n="<<&n<<endl;
        system("pause"); return 0;
}// end main
```

اسناد 23 كقيمة للمتغير n

طباعة قيمة المتغير n

طباعة عنوان المتغير n

يعطي هذا البرنامج على خرجه:

```
the value of n is: n=23
the address of n is: &n=010FEFA8
Press any key to continue . . .
```

References

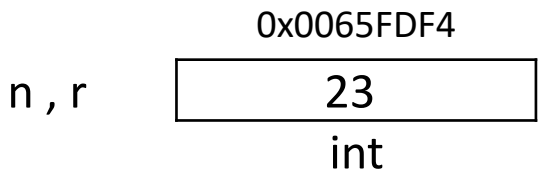
المراجع 1

- المرجع: هو اسم آخر مرادف لاسم المتغير، يعلن عنه باستخدام المعامل المرجعي & والذي يلحق بنوع المرجع.

- التصريح عن المرجع
- التصريح التالي:

```
int n = 23 ;
int & r = n ; // r is a reference for n ;
```

✓ الإعلان عن متغير صحيح اسمه n وخصصت له قيمة ابتدائية مقدارها 23.
✓ أعلن عن متغير r بأنه مرجع لـ n:



✓ إن قيمة r هي ذاتها قيمة n وأي تغير في قيمة n سيصيب قيمة r والعكس صحيح.
✓ إن n و r لهما نفس العنوان في الذاكرة.
✓ إن n و r هما إسمان رمزيان لنفس المكان في الذاكرة.

References

المراجع 2

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void main()
{       int n=23;
        int &r=n;// r is reference for n
        cout<<"the value of n is: n="<<n<<endl;
        cout<<"the value of r is: r="<<r<<endl;
        cout<<"&n="<<&n<<endl;
        cout<<"&r="<<&r<<endl;
        cout<<endl;
        --n;
        cout<<"the value of n is: n="<<n<<endl;
        cout<<"the value of r is: r="<<r<<endl;
        cout<<endl;    r*=2;
        cout<<"the value of n is: n="<<n<<endl;
        cout<<"the value of r is: r="<<r<<endl;
        system("pause"); //return 0;
}// end main
```

طباعة قيمة المتغير n

طباعة قيمة المتغير r

طباعة عنوان المتغير n

طباعة عنوان المتغير r

إنقاص قيمة المتغير n بمقدار 1

طباعة قيمة كل من n,r
بعد إنقاص n بقدار 1

طباعة قيمة كل من n,r
بعد مضاعفة قيمة r

Pointers

المؤشرات 1

- المؤشرات: هي عبارة عن متغيرات تحتوي على عناوين والتي تتضمن قيم مخزنة ضمنها في الذاكرة .

✓ إن المتغير يدل بشكل مباشر على قيمة.
✓ يدل المؤشر بشكل غير مباشر على قيمة.
✓ تسمى عملية الدلالة على قيمة بواسطة مؤشر بالعملية غير المباشرة indirection.

- يمكن التصريح عن المؤشر من خلال الصيغة العامة التالية:

```
type *pointerName ;
```

✓ حيث أن:
- type: نوع القيمة التي يشير لها المؤشر.
- pointerName: اسم المؤشر.
- المعامل * هو معامل المؤشر.
- عندما يستخدم بالشكل السابق فإنه يعلن أن مابعده متغير من نوع مؤشر يشير إلى قيمة من النوع الذي قبله.

Pointers

المؤشرات 2

- التصريح :

```
int * x ;
```

✓ يعلن أن x مؤشر لرقم صحيح.

- التصريح:

```
int * x , y ;
```

✓ يعلن أن x مؤشر لرقم صحيح وأن y متغير صحيح.

- التصريح:

```
int * x , * y ;
```

✓ يعلن أن كل من x و y مؤشران لرقمين صحيحين.

- معاملات المؤشرات هنالك مؤثران خاصان بالمؤشرات هما *و &.

✓ المؤثر & هو مؤثر أحادي يرجع عنوان الذاكرة لمعامله.
✓ المؤثر * يرجع مايدل إليه العنوان من الذاكرة.

Pointers

المؤشرات 3

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void main()
{       int n=23;
        int *p=&n;
        cout<<"the value of n is : n="<<n<<endl;
        cout<<"the address of n is : &n="<<&n<<endl;
        cout<<"the value of p is : p="<<p<<endl;
        cout<<"the address of n is : &p="<<&p<<endl;
        system("pause"); //return 0;
}// end main
```

p holds the address of n

طباعة قيمة n من الذاكرة

طباعة عنوان تخزين n في الذاكرة

طباعة قيمة p من الذاكرة

طباعة عنوان تخزين p في الذاكرة

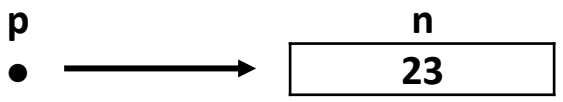
```
the value of n is : n=23
the address of n is : &n=0093F008
the value of p is : p=0093F008
the address of n is : &p=0093F00C
Press any key to continue . . .
```

إن عنوان تخزين n موضوع في p
وبالتالي هو قيمة p
المؤشر p محجوز له عنوان يخزن ضمنه قيمته
وهو 4byte زياده لعنوان n كونه صحيح

- إن `int * p = & n;` سيخزن عنوان n في المكان المحجوز لـ p .
- نلاحظ من خرج البرنامج أن المتغير n استخدم العنوان 0x0093F008 لتخزين القيمة 23 وسيملك p قيمة 0x0093F008 .

يمكن تصور المتغيرين n و p كما في التخطيط التالي:

- إذاً، قيمة المؤشر هي عنوان ويعتمد على المطابق ونظام التشغيل للحاسب الذي يجري عليه البرنامج.
- في معظم الحالات تكون القيمة الفعلية لهذا العنوان غير مهمة للمبرمج ولذلك فإن التخطيط السابق عادة ما يتم رسمه كما في الشكل:



Pointers

المؤشرات 5

- المؤشر هو محدد وضع حيث يبين أين توجد قيمة أخرى.
- يستعمل المؤشر وحده للحصول على القيمة التي يشير إليها.
- معامل العنوان & ومعامل إعادة المرجعية * عاملان متتامان حيث أن: n = *p (1) عندما p = &n (2)، بتبديل 2 في 1 نحصل على n =*&n وأيضاً بتبديل 1 في 2 نحصل p = &*p .
- المعامل * معامل متمم للمعامل & وهو معامل أحادي يرجع قيمة المتغير الموجود في العنوان الذي يشكل معامله.

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void main()
{
int n=23;
int *p=&n;
int &r=*p;
cout<<"*p="<<*p<<endl;
cout<<"r="<<r<<endl;
  system("pause"); //return 0;
}// end main
```

ستطبع قيمة المتغير n وهي القيمة التي يشير لها العنوان الموجود في المؤشر p.

r مرجع للقيمة التي يشير إليها p، وبالتالي تتحدد قيمة r بالقيمة التي يشير إليها p وهي قيمة n.

The Pointer's Expressions

1 تعابير المؤشرات

- تستخدم المؤشرات التعابير الحسابية وتعابير الإسناد والمقارنة.
- المساواة الإسنادية للمؤشرات:
  - ✓ استخدام المؤشر كطرف أيمن في العلاقات الإسنادية لإسناد قيمته إلى مؤشر آخر.
  - ✓ إن المؤشرين يجب أن يشيرا إلى معطيات من نفس النمط.

كمثال:

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void main()
{       int n=23;
        int *p1,*p2;
        p1=&n;
        p2=p1;
        cout<<"&n="<<&n<<endl;
        cout<<"p1="<<p1<<endl;
        cout<<"p2="<<p2<<endl;
        system("pause");
}// end main
```

اسناد عنون المتغير n الى المؤشر p1.

اسناد قيمة المؤشر p1 الى المؤشر p2.

ستكون قيم الخرج الثلاث متساوية

The Pointer's Expressions

تعابير المؤشرات 2

كمثال بخرج التابع sizeeof() الحجم المججوز

sizeeof() يستخدم مع المعطيات الأولية والمعرفة والكائنات

```
 #include "stdafx.h“
#include<iostream>
using namespace std;
void main()
{cout<<"number of bytes used:\n";
cout<<"\t char :"<<sizeof(char)<<endl;
cout<<"\t short :"<<sizeof(short)<<endl;
cout<<"\t int :"<<sizeof(int)<<endl;
cout<<"\t long :"<<sizeof(long)<<endl;
cout<<"\t unsigned short :"<<sizeof(unsigned short)<<endl;
cout<<"\t unsigned char :"<<sizeof(unsigned char)<<endl;
cout<<"\t unsigned int :"<<sizeof(unsigned int)<<endl;
cout<<"\t unsigned long :"<<sizeof(unsigned long)<<endl;
cout<<"\t signed char :"<<sizeof(signed char)<<endl;
cout<<"\t float :"<<sizeof(float)<<endl;
cout<<"\t double :"<<sizeof(double)<<endl;
cout<<"\t long double :"<<sizeof(long double)<<endl;
system("pause");    }// end main
```

The Pointer Arithmetic

العمليات الحسابية على المؤشرات

- تطبق عمليات الزياده والنقصان للقيم الصحيحة (++، --، =+، +، -).
- طرح أو جمع مؤشين من أو على بعضهما.
- زيادة أو نقصان المؤشر بـ 1 ستزيد أو تنقص قيمة هذا المؤشر بمقدار حجم النوع الذي يشير إليه المؤشر.

يمكن معاينة جميع هذه العمليات من خلال المثال التالي :

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void main()
{       int n=23;       int *p=&n;
        cout<<"&n="<<&n<<" and p="<<p<<endl;
        ++p;    cout<<"after p++ :  p="<<p<<endl;
        p+=2;   cout<<"after p+=2:  p="<<p<<endl;
        p--;    cout<<"after p-- :  p="<<p<<endl;
        p-=5;   cout<<"after p-=5:  p="<<p<<endl;
        int *p1=&n;     cout<<"p1="<<p1<<endl;
        cout<<"p1-p="<<p1-p<<endl;
        system("pause");
}// end main
```

طباعة عنوان n ومحتوى p وهي القيمة 0x009BED64

طباعة محتوى p بعد الزياده بـ 1 لتصبح 0x009BED68

طباعة محتوى p بعد الزياده بـ2 لتصبح 0x009BED70

طباعة محتوى p بعد النقصان بـ1 لتصبح 0x009BED6c

اسناد عنوان n للمؤشر p1 وطباعته 0x009BED64

طباعته ناتج طرح p1-p والمساويه 3 أي بعد تحويله للنوع

العلاقة بين المؤشرات والمصفوفات 1

The Relation Between Pointers And Arrays

- اسم المصفوفة ليس إلا مؤشر صحيح يدل لبدايتها والعكس صحيح.
- يمكن استخدام أحدها للحلول مكان الآخر لعمل عمل الآخر.
- التصريح int a[4] يدل لمصفوفه اسمها a عدد عناصرها 4 ومن النمط الصحيح.

- التصريحين متكافئتيب عنوان العنصر الأول في الصف إلى المؤشر p:

```
int *p = a;
int *p = &a[0];
```

- أضف إلى ذلك فإن شيفرة استدعاء التابع بالصف : int * a ↔ int a[ ]

- يمكن الوصول إلى العنصر الرابع في الصف a[3] باستخدام التعبير: cout << * (p+3) ;

- أن * (p+3) هي القيمة التي يشير إليها المؤشر p بعد زيادته بمقدار 3 .

- إذا أشر مؤشر إلى بداية الصف فإن الانزياح المضاف إليه يدل على أحد عناصر الصف الذي يطابق دليله قيمة ذلك الانزياح.
- ويجب التأكيد على وجود القوسين في الكتابة السابقة لأن أولوية العملية * هي أعلى من أولوية الجمع.
- العبارة السابقة دون أقواس أي p + 3 * فإن ذلك يعني إضافة 3 إلى العنصر a[0]

العلاقة بين المؤشرات والمصفوفات 2

The Relation Between Pointers And Arrays

- إن : p+3 ↔ & a[3]
- يستخدم إسم الصف في العمليات الحسابية فمثلاً التعبير: (a + 3) * يعطي قيمة العنصر [3]a من عناصر الصف.
- يضاف الدليل للمؤشرات حيث أن التعبير: [1]p يدل على العنصر[1]a.
- اسم الصف عبارة عن مؤشر ويستحسن أن يكون ثابت وهو يشير دوماً إلى بداية الصف ولذلك فإن التعبير: a + = 3 هو تعبير غير محبب.
- المصفوفة ثنائية البعد ينظر لها مصفوفة على مصفوفة.
- يمكن الحصول على عنوان أول عنصر في مصفوفة متعددة الأبعاد من خلال التعليمة: cout<<*(a);
- يمكن الحصول على عنوان العنصر a[i][j] من خلال التعليمة:

```
cout<<*(a)+(m*i)+j;
```

- إن m هو عدد أعمدة المصفوفة ثنائية البعد،
- إن عناصر المصفوفة ثنائية البعد تخزن في الذاكرة في أماكن متجاورة كما لوكانت عناصر تلك المصفوفة تنتمي لصف عدد عناصره هو m*n حيث أن n هو عدد الأسطر في المصفوفة ثنائية البعد.

- قيمة عنصر المصفوفة الثنائية البعد a[i][j] يمكن أن نحصل عليها من خلال التعليمة:

```
cout<<*(*(a)+(m*i)+j);
```

The Relation Between Pointers And Arrays

العلاقة بين المؤشرات والمصفوفات 3

المثال التالي يوضح هذه العلاقة مصفوفة أحادية البعد :

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void main()
{int a[]={5,7,9,11};
 int *p=a;
 cout<<"a array is:"<<endl;
 for(int i=0;i<4;i++)cout<<"a["<<i<<"]="<<a[i]<<endl;
 cout<<endl<<"pointer/offset notation where"<<endl;
 cout<<"the pointer is the array name"<<endl;
 for(int j=0;j<4;j++)cout<<"*(a+"<<j<<")="<<*(a+j)<<endl;
 cout<<endl<<"pointer subscript notation"<<endl;
 for(int i=0;i<4;i++)cout<<"p["<<i<<"]="<<p[i]<<endl;
 cout<<endl<<"pointer/offset notation"<<endl;
 for(int j=0;j<4;j++)cout<<"*(p+"<<j<<")="<<*(p+j)<<endl;
 system("pause");
}// end main
```

طباعة محتوى المصفوفة باستخدام اسمها ودليلها

طباعة محتوى المصفوفة باستخدام اسمها كمؤشر

طباعة محتوى المصفوفة باستخدام مؤشر كإسم لها

طباعة محتوى المصفوفة باستخدام مؤشر لها

العلاقة بين المؤشرات والمصفوفات 4

نتائج المثال :

```
a array is:
a[0]=5
a[1]=7
a[2]=9
a[3]=11

pointer/offset notation where
the pointer is the array name
*(a+0)=5
*(a+1)=7
*(a+2)=9
*(a+3)=11

pointer subscript notation
p[0]=5
p[1]=7
p[2]=9
p[3]=11

pointer/offset notation
*(p+0)=5
*(p+1)=7
*(p+2)=9
*(p+3)=11
Press any key to continue . . .
```

طباعة محتوى المصفوفة باستخدام اسمها ودليلها

طباعة محتوى المصفوفة باستخدام اسمها كمؤشر

طباعة محتوى المصفوفة باستخدام مؤشر كإسم لها

طباعة محتوى المصفوفة باستخدام مؤشر لها

العلاقة بين المؤشرات والمصفوفات 4

المثال التالي يوضح هذه العلاقة مصفوفة ثنائية البعد :

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void main()
{const int n=2;     const int m=3;
int a[n][m];        int i,j;
cout<< "enter a ["<<n<<"]["<<m<<"]"<<endl;
for(i=0;i<n;i++)
   for(j=0;j<m;j++)          cin>>a[i][j];
cout<<"a array is:"<<endl;
for(i=0;i<n;i++)
{         for(j=0;j<m;j++) cout<<a[i][j]<<" ";    cout<<endl;      }
for(i=0;i<n;i++)
{         for(j=0;j<m;j++)
             cout<<*(a)+(m*i)+j<<"  "<<*(*(a)+(m*i)+j)<<endl;     }
system("pause");
}// end main
```

طباعة عناصر المصفوفة باستخدام اسمها

طباعة عناوين عناصر المصفوفة

طباعة عناصر المصفوفة

العلاقة بين المؤشرات والمصفوفات 4

نتائج مثال مصفوفة ثنائية البعد :

```
enter a [2][3]
11 22 33
44 55 66
a array is:
11 22 33
44 55 66
0133F374  11
0133F378  22
0133F37C  33
0133F380  44
0133F384  55
0133F388  66
Press any key to continue . . .
```

إدخال عناصر المصفوفة

طباعة عناصر المصفوفة باستخدام اسمها

طباعة عناصر المصفوفة وعناوينها

إعادة مؤشر من التابع 1

Returnning Pointer From Function

من الممكن أن تكون القيمة العائدة من التابع مؤشراً عندئذ لابد أن يتم التصريح عن التابع على النحو التالي :

type *function_name(parameter_lists)

يقوم البرنامج التالي بالبحث عن سلسلة ضمن سلسلة أخرى ويعيد تابعه loc مؤشراً على مكان وجود السلسلة الجزئية ضمن الأخرى.

```
#include "stdafx.h"
#include<iostream>
using namespace std;
int*loc(int*a1,int *a2,int n1,int n2)
       {int *endl=a1+n1;
              for(int *p1=a1;p1<endl;p1++)
               if(*p1==*a2)
                     { int j;
                     for( j=0;j<n2;j++)
                            if(p1[j]!=a2[j])break;
                            if(j==n2)return p1;
                     }
              return 0;
       }
```

إعادة مؤشر من التابع 1

```
void main()
{
	int a1[9]={11,11,11,11,11,22,33,44,55};
	int a2[5]={11,11,11,22,33};
	cout<<"array a1 begins at location\t"<<a1<<endl;
	cout<<"array a2 begins at location\t"<<a2<<endl;
		int *p=loc(a1,a2,9,5);
	if(p){
		cout<<"array a2 found at location\t"<<p<<endl;
		for(int i=0;i<5;i++)
		cout<<"\t"<<&p[i]<<":"<<p[i]<<"\t"<<&a2[i]<<":"<<a2[i]<<endl;
		}
	else cout<<"not found.\n";
system("pause");
}// end main
```

إعادة مؤشر من التابع 2

نتيجة تنفيذ التمرين:

```
array a1 begins at location     00F9EFD4
array a2 begins at location     00F9EFF8
array a2 found at location      00F9EFDC
        00F9EFDC:11     00F9EFF8:11
        00F9EFE0:11     00F9EFFC:11
        00F9EFE4:11     00F9F000:11
        00F9EFE8:22     00F9F004:22
        00F9EFEC:33     00F9F008:33
Press any key to continue . . .
```

إستخدام المؤشرمع المكدس 1

مثال يوضح كيفية الاستفادة من stack ومبدأ LIFO في عملية تحويل العدد من النظام العشري إلى النظام الثنائي:

```
#include "stdafx.h"
#include<iostream>
#include<stdlib.h>
using namespace std;

const int stacksize=50;
int*p ;  int stack[stacksize];

void push (int i){ if (p==(stack+stacksize))
{ cout<<"stack over flow"; exit(1); }
*p=i; p++;
}

void pop(int &i) {if(p==stack)
{cout<<"stack under flow";exit(1);}
p--;i=*p;
}
```

LIFO

إستخدام المؤشرمع المكدس 1

```
void main()
{
int x;int i,n=0;
while(1)
{ cout <<"enter positive number:"; cin >>x; cout<<endl;
if (x<0) break;
p=stack;
if (x==0)cout<<"0";
while (x!=0)
{      i=x%2;        push(i);     n++;   x=x/2;}
while (n>0)
{ pop(i);     cout<<i<<"   ";n--;}
cout<<endl;   }
system("pause");
}// end main
```

إستخدام المؤشرمع المكدس 1

سنحصل على النتيجة التالية:

```
enter positive number:23

1 0 1 1 1
enter positive number:123

1 1 1 1 0 1 1
enter positive number:2444

1 0 0 1 1 0 0 0 1 1 0 0
enter positive number:-1

Press any key to continue . . .
```

إستدعاء التابع بالمؤشر

Calling Function By Pointer

ذكرنا سابقاً طرق استدعاء التابع وذكرنا أنه يمكن استدعاء التابع بالعنوان ولهذا الاستدعاء نوعان الاستدعاء بالمرجع والاستدعاء بالمؤشر.

عند استدعاء التابع بالمؤشر يجب على التابع الذي يتلقى عنواناً لوسيط فعلي أن يتضمن في تعريفه مؤشراً كوسيط شكلي.

على سبيل المثال ، ليكن البرنامج التالي :

```
#include<iostream.h>
void cube(int *p) {          *p=*p* *p* *p; }

void main()
{          int number=5;     cube(&number);
           cout<<"5 power 3 = "<<number<<endl;
}
```

في هذا المثال يتضمن إعلان التابع cube المؤشر int * p الذي يتلقى عنوان المتغير الفعلي number (& number) الذي خصصت له القيمة 5 ويتم داخل التابع cube تم تغيير القيمة التي يشير إليها المؤشر إلى 125 أي 5*5*5 لذلك أصبح p يشير إلى القيمة 125 وطالما أن p يشير إلى المتغير number لذلك أصبحت قيمة number هي 125.

لنحصل على هذه القيمة لا بد من طباعة number بعد استدعاء التابع cube .

إستخدام الواصف const مع المؤشرات 1

Using The Const Qualifier With Pointers

إذا سبق الواصف const متغير محدد ما وعند المحاولة لتغيير قيمته فإن المترجم يلتقط هذه المحاولة ويمنع القيام بها مرسلاً إما رسالة تحذير أو خطأ حسب نوع هذا المترجم، هذا ويمكن استخدام المؤشر مع const في أربعة أشكال:

1- مؤشر غير ثابت لمعطيات غير ثابتة.

نحصل على أعلى مستوى للوصول (أي يمكن تغيير المعطيات باستخدام المؤشرات عليها ويمكن أيضاً تغيير وجهة المؤشر ليؤشر على معطيات أخرى).لا يتضمن التصريح عن المؤشر غير الثابت على المعطيات غير الثابتة على الواصف const .

2- مؤشر غير ثابت لمعطيات ثابتة.

تستخدم للتأشير على معطيات ثابتة ذات نمط معين ويمكن تغييرها لتؤشر على معطيات أخرى لها نفس النمط ولكن لا يمكن تغيير المعطيات المؤشر عليها. const int * p;

3- مؤشر ثابت لمعطيات غير ثابتة.

يقوم بالتأشير دوماً على نفس موضع الذاكرة الذي يمكن تغيير المعطيات المخزنة فيه باستخدام المؤشر ويذكرنا هذا النوع من المؤشرات بأسماء المصفوفات التي هي عبارة عن مؤشرات ثابتة على أول عنصر من عناصرها، ويمكن الوصول إلى كافة عناصر المصفوفة وتغييرها باستخدام اسم المصفوفة ودليل هذه العناصر.int * const p;

4- مؤشر ثابت لمعطيات ثابتة.

تؤمن أقل إمكانيات الوصول للمعطيات، تؤشر المؤشرات السابقة دوماً على نفس الموضع من الذاكرة وتبقى المعطيات في ذلك الموضع على حالها دون تغيير، يستخدم هذا النوع من المؤشرات لتمرير مصفوفة إلى تابع يقوم باستعراض عناصرها بالاعتماد على أدلة العناصر ولا يستطيع تغيير قيم هذه العناصر كما هو الحال عند طباعة عناصر المصفوفة أو البحث عن عنصر ضمن مصفوفة

const int * const p;

إستخدام الواصف const مع المؤشرات 2

Using The Const Qualifier With Pointers

يمكن إيضاح استخدام الواصف const مع المؤشرات من خلال المثال الشامل التالي الذي يتضمن الأشكال الأربعة وقد نفذت التعبيرات الممكنة عليها أما التعبيرات الخاطئة فقد كتبت كتعليق :

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void main() {
    int x =23;int *p =&x;
    cout<<"x="<<x<<" and *p="<<*p<<endl;
    cout<<"&x="<<&x<<" and p="<<p<<endl;
    cout<<" p is non_constant pointer to non-constant data"<<endl;
    cout<<"which is x so p and x can be changed."<<endl;

    ++x;++p;
    cout<<"after x++ : x="<<x<<endl; cout<<"after p++ : p="<<p<<endl<<endl;

        int * const cp=&x;
              cout<<"x="<<x<<" and cp="<<cp<<endl;
        cout<<" cp is constant pointer to non-constant data"<<endl;
        cout<<"which is x so  only x can be changed ."<<endl;
```

إستخدام الواصف const مع المؤشرات 3

```
  ++x;
  //++cp;error C2166:l-value specifies const object
   cout<<"after x++ : x="<<x<<"\n\n\n";

 const int y=44; const int *pc=&y;
 cout<<"y="<<y<<" and pc="<<pc<<endl;
 cout<<" pc is non_constant pointer to constant data"<<endl;
 cout<<"which is y so only pc can be changed."<<endl;

 pc++;       //(*pc)++;error C2166:l-value specifies const object
 cout<<"after pc++ :pc="<<pc<<endl;
 cout<<endl;

 const int* const cpc=&y;
     cout<<"y="<<y<<" and cpc="<<cpc<<endl;
 cout<<" cpc is constant pointer to constant data"<<endl;
 cout<<"which is y so  cpc and y can not be changed ."<<endl;
   //++(*cpc);error C2166:l-value specifies const object
     //++cpc;error C2166:l-value specifies const object
 system("pause");
}// end main
```

Using The Const Qualifier With Pointers

إستخدام الواصف const مع المؤشرات 5

```
x=23 and *p=23
&x=00F3EE8C and p=00F3EE8C
 p is non_constant pointer to non-constant data
which is x so p and x can be changed.
after x++ : x=24
after p++ : p=00F3EE90

x=24 and cp=00F3EE8C
 cp is constant pointer to non-constant data
which is x so  only x can be changed .
after x++ : x=25

y=44 and pc=00F3EE78
 pc is non_constant pointer to constant data
which is y so only pc can be changed.
after pc++ :pc=00F3EE7C

y=44 and cpc=00F3EE78
 cpc is constant pointer to constant data
which is y so  cpc and y can not be changed .
Press any key to continue . . .
```

توابع التخصيص الديناميكي

Dynamic Allocation Functions

حالما تتم عملية الترجمة للبرامج المكتوبة بلغة C++، تقوم هذه البرامج بتنظيم ذاكرة الحاسب إلى أربع مناطق:

- ✓ منطقة تحمل البرنامج المكتوب (program code).
- ✓ منطقة تحمل المتحولات الشاملة (global variables).
- ✓ منطقة تعمل كمكدس(stack).
- ✓ منطقة تعمل كمكوم (heap).

والمكوم هو المساحة من الذاكره الحره والتي يتم إدارتها بواسطة توابع التخصيص الديناميكي delete و new.

يقوم التابع new بتخصيص صريح لذاكرة المؤشر نفسه، فهو يعيد في المؤشر عنوان مجموعة من s بايت غير مخصصة بالذاكرة، حيث s هو حجم النوع الذي يشير إليه المؤشر، تنسيب هذا العنوان للمؤشر يضمن أن القيمة التي يشير إليها المؤشر غير مستخدمه بواسطة متغير أخر.

يستخدم التابع new كمايلي:

```
p=new int;        *p=3;
```

حيث تم بداية الإعلان عن مؤشر إلى النوع الصحيح اسمه p.

يقوم new int بتخصيص عنوان 4 بايت من الذاكرة الحره (حجم النوع الصحيح) إلى المؤشر p، هذا التخصيص يمكننا من تحميل القيمة 3 في العنوان السابق.

الأسطر الثلاثة السابقة يمكن كتابتها بالشكل:

```
int*p=new int;  *p=3;
```

كذلك هذين السطرين يمكن أن يكتبا بالشكل:

```
int *p=new int(3);
```

مشاكل المؤشرات

Problems with pointer

- لا شيء يسبب المتاعب كاستخدام المؤشر الذي لم يعط قيمة ابتدائية ويمكننا القول أن للمؤشرات مزيج من الميزات الحسنة والسيئة؟
- فبينما تمدنا المؤشرات بطاقة عظيمة وهي ضرورية في أكثر البرامج إلا أن القيمة الخاطئة للمؤشر من الصعب جداً كشفها.
- ليس المؤشر بحد ذاته مشكلة, بل المشكلة هي أنه في كل مرة تنجز فيها عملية ما على المؤشر فإننا نقرأ أو نكتب في جزء ما من الذاكرة؟
- إذا كنا نقرأ منه فأسوأ ما يمكن أن يحدث هو أن نحصل على قيمة عديمة المعنى.
- عندما نكتب فقد نكتب فوق جزء آخر من برنامجنا أو معطياتنا.
- وهذا لا يظهر إلا بعد تنفيذ البرنامج وقد يتطلب منا معرفة مكان الخطأ كما ذكرنا سابقاً.
- من أكثر الأخطاء الناجمة عن استخدام المؤشرات شيوعاً ما يلي:

- الخطأ الناجم عن عدم تهيئة المؤشر(uninitialized pointer).
- الخطأ الناجم عن عدم فهم كيفية استخدام المؤشرات هذا موضح في البرنامج التالي.

Problems with pointer

مشاكل المؤشرات

1- الخطأ الناجم عن عدم تهيئة المؤشر(uninitialized pointer).

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void main()
{       int x,*p;                x=10;          *p=x;          cout<<*p<<endl;
        system("pause");
}// end main
```

إن الإعلان int *p قام بتخصيص ذاكرة للمؤشر p فقط, قيمة المؤشر ستكون عنوان بالذاكرة, ولكن الذاكرة في هذه اللحظة لم يتم تخصيصها, هذا يعني أن عنوان الذاكرة هذا قد يكون مخصص بواسطة متغير أخر, لذلك عند تنفيذ البرنامج سيعطي المترجم تحذيراً بأن المؤشر استخدم بدون إعطاءه قيمة ابتدائية, ولن ينفذ البرنامج حيث سيتم إعطاء رسالة مفادها أن البرنامج قام بعملية غير شرعية وسيتم إغلاقه وفق نوع المطابق وقد تكون من قبيل .

An unhandled exception of type 'System.NullReferenceException' occurred in pointerAndString.exe

Additional information: Object reference not set to an instance of an object.

لتصحيح البرنامج لابد من كتابة العبارة التالية بعد الإعلان عن x وp:

p=&x;

2- الخطأ الناجم عن عدم فهم كيفية استخدام المؤشرات هذا موضح في البرنامج التالي.

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void main()
{       int x,*p;       x=10;   p=x; // p=%&x
        cout<<*p<<endl;
        system("pause");
}// end main
```

إلبرنامج واضح أن المقصود منه هو طباعة قيمة x, لكنه لن ينفذ لأن المترجم سيعطي خطأ وهو عدم إمكانية التحويل من النمط الصحيح إلى النمط مؤشر للنمط الصحيح ورسالة الخطأ هي التالية:

```
error C2440: '=' : cannot convert from 'int' to 'int *'
Conversion from integral type to pointer type requires reinterpret_cast, C-style cast or
function-style cast
```

هذا البرنامج ينفذ بشكل صحيح ويطبع القيمة 10 إذا تم الإسناد قبل الطباعة وليس ما بعد التعريف مباشرتاً كما في الحاله السابقة:

p=&x;

بعض الآسئله للحل

1. مثال يوضح تطبيق مفيد لصف المؤشرات, ويمكن من ترتيب صف بطريقة غير مباشرة بتغيير المؤشرات للعناصر بدلاً من تحريك العناصر نفسها.

2. استخدام صف من المؤشرات للتوابع: أنه يمكن أن يطلب من مستخدم اختيار أحد الخيارات ضمن قائمة ما (يمكن اختيار من 1 حتى 5 خيارات), ويتعامل كل خيار من الخيارات السابقة مع تابع محدد, لإنجاز ذلك يمكن استخدام مؤشر على كل تابع من التوابع المعرفة, وتخزن ضمن مصفوفة من المؤشرات على توابع, ويستفاد من خيار المستخدم على أنه دليل عنصر المصفوفة المطلوب والذي هو عبارة عن مؤشر على التابع المطلوب استدعاؤه.

3. تطوير تمرين إستخدام المؤشرمع المكدس ليتعامل مع مختلف أمظمة العد مثل رباعي، ثماني، سداسي عشر.

إستخدام الواصف const مع المؤشرات 1

Non-constant pointer to non-constant data

الشكل العام int * p;

(أي غياب الواصفة نستطيع تغيير القيمة وتغيير موضع المؤشر)

```
#include "stdafx.h"
#include<iostream>
using namespace std;

void changetopositive(int *p)
{for(int i=1;i<=6;i++)    {      *p=-1*(*p);  ++p;   }      }

void main(){ int a[]={-5,-6,-7,-8,-9,-10};int i; cout<<"a array : ";
for( i=0;i<6;i++) cout<<a[i]<<" ";    cout<<endl;
changetopositive(a);
cout<<"a array after the change is :"<<endl;
for(i=0;i<6;i++)    cout<<a[i]<<"  ";   cout<<endl;
system("pause");    }// end main
```

```
a array : -5  -6  -7  -8  -9  -10
a array after the change is :
5  6  7  8  9  10
Press any key to continue . . .
```

إستخدام الواصف const مع المؤشرات 2

non_constant pointer to constant data

الشكل العام const int * p;

(أي لانستطيع تغيير القيمة ونستطيع تغيير موضع المؤشر)

```
#include "stdafx.h"
#include<iostream>
using namespace std;

void changetopositive (const int *p)
{for(int i=1;i<=6;i++)    {     *p=-1*(*p);  ++p;   }     }

void main(){ int a[]={-5,-6,-7,-8,-9,-10};int i; cout<<"a array : ";
for( i=0;i<6;i++) cout<<a[i]<<"  ";    cout<<endl;
changetopositive(a);
cout<<"a array after the change is :"<<endl;
for(i=0;i<6;i++)    cout<<a[i]<<"   ";   cout<<endl;
system("pause");    }// end main
```

سنحصل على رسالة الخطأ التالية:

error C3892: 'p' : you cannot assign to a variable that is const

إستخدام الواصف const مع المؤشرات 3

non_constant pointer to constant data

(يمكن استخدام النوع السابق من المؤشرات مع الصفوف التي يراد تمريرها إلى تابع يقوم بمعالجة كل عنصر من عناصرها دون تغيير قيمتها, مثل طباعة عناصر المصفوفة, البحث عن عنصر فيها)

```
#include "stdafx.h"
#include<iostream>
using namespace std;

void printarray(const int *p)
{for(int i=1;i<=6;i++)    {cout<<*p<<"  ";    p++;  }
              cout<<endl;
}
void main()
{      int a[]={-5,-6,-7,-8,-9,-10};
       cout<<"a array is : ";    printarray(a);      system("pause");
}// end main
```

سنحصل على النتيجة التالية:

```
a array is : -5  -6  -7  -8  -9  -10
Press any key to continue . . .
```

إستخدام الواصف const مع المؤشرات 4

constant pointer to non-constant data

الشكل العام int *const p; (أي نستطيع تغيير القيمة ولانستطيع تغيير موضع المؤشر)

```
#include "stdafx.h"
#include<iostream>
using namespace std;
 void changetopositive( int* const p)
{            for(int i=0;i<6;i++)      *(p+i)=-*(p+i);}
void main()
{     int i, a[]={-5,-6,-7,-8,-9,-10};       cout<<"a array is : ";
for( i=0;i<6;i++)cout<<a[i]<<"  ";  cout<<endl;
changetopositive(a);      cout<<"a array after the change is : ";
for(i=0;i<6;i++)cout<<a[i]<<"  ";cout<<endl;  system("pause");
}// end main
```

لا يمكن استخدام p++وسنحصل على النتيجة التالية :

```
a array is : -5  -6  -7  -8  -9  -10
a array after the change is : 5  6  7  8  9  10
Press any key to continue . . .
```

إستخدام الواصف const مع المؤشرات 5

Const pointer to const data

الشكل العام const int *const p; (أي لانستطيع تغيير القيمة ولانستطيع تغيير موضع المؤشر)

```
#include "stdafx.h"
#include<iostream>
using namespace std;
void printarray(const int * const p)
{ for(int i=0;i<6;i++) {  cout<<*(p+i)<<"  "; }          cout<<endl;
}
void main()
{int a[]={-5,-6,-7,-8,-9,-10};
       cout<<"a array is : ";
       printarray(a);
       system("pause");
}// end main
```

سنحصل على النتيجة التالية:

```
a array is : -5  -6  -7  -8  -9  -10
Press any key to continue . . .
```

جَامعة
المَنارة
MANARA UNIVERSITY

انتهت محاضرة الأسبوع 1