



كلية الهندسة – قسم المعلوماتية

مقرر برمجة 2

ا.د. علي سليمان

Operator overloading1

المحاضرة الرابعة

الفصل الثاني 2025-2026

1. Introduction.
 2. Operator overloading, concept and implementation.
 3. Restrictions on operator overloading.
 4. Operator overloading using methods as Class Members vs. as friend methods
 5. Overloading Stream-Insertion and Stream-Extraction Operators,
 6. Overloading Unary, Binary Operators.
 7. Overloading ++ and --.
 8. Overloading new and delete
 9. Case Study,
1. مقدمة
 2. التحميل الزائد للمعاملات، مفهومه وتحقيقه.
 3. القيود على التحميل الزائد للمعاملات.
 4. التحميل الزائد للمعاملات باستخدام التوابع الأعضاء والصديقة.
 5. التحميل الزائد للمعاملات الدخلى (الحشر) والخرج (الاستخراج).
 6. التحميل الزائد للمعاملات الاحادية والثنائية,
 7. التحميل الزائد للمعاملات ++ , -- .
 8. التحميل الزائد للمعاملين new & delete
 9. دراسة حالة.

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

- تستخدم الأصناف في إنشاء أنماط بيانات معرفة من قبل المستخدم.
- استخدام هذه الأنماط من خلال اشتقاق أغراض (التصريح عن متحولات) منها.
- إن التعامل مع هذه الأغراض يتم من خلال إرسال رسائل لها (على شكل استدعاءات للتوابع الأعضاء).
- للتعامل مع أغراض من الصنف Time، كان بإمكاننا استدعاء التوابع الأعضاء setTime، setHour، setMinute، getHour، getMinute، printStandard،..... إلخ.
- تعميم: يمكن للمبرمج تضمين الصنف الذي يقوم بتعريفه كافة التوابع التي يرغب بأن تتعامل مع أغراض من هذا النمط، ومن ثم تحقيق هذا التعامل من خلال استدعاء هذه التوابع مع الأغراض التي يقوم بتعريفها.
- إن هذا يتطلب من المبرمج خلال مرحلة التصميم التحري عن كامل العمليات التي تنطوي عليها عمليات استثمار الصنف وتعريف توابع أعضاء، أصدقاء، مستقلة للصنف لتحقيقها.
- لو رغبتنا أن يتضمن تعريف الصنف Time إمكانية إجراء العمليات الحسابية (الجمع والطرح) وعمليات الإدخال والإخراج والمقارنة لقيم الأغراض من النمط Time، لكان بالإمكان أن يتم تعريف الصنف Time على النحو التالي:

Introduction



مقدمة 2

```
// OperatorOverloading85.cpp : main project file.
```

```
#include <iostream>
#include <iomanip>
using namespace std;
class Time {public:
    Time() ; //constructor
    void Time_IN(); //entering Time objects
    void Time_OUT(); //printing Time objects
    Time add(Time,Time) ; //finding sum of Time objects
    Time sub(Time,Time); //finding sub of time objects
    bool is_greater(Time);
private:
    int hour; // 0 - 23 (24-hour clock format)
    int minute; // 0 - 59
    int second; // 0 - 59
    char sep; //:
}; // end clas Time
```

Introduction



مقدمة 3

```
Time::Time()
{   hour=minute=second=0;       sep=': '; } // end Time constructor

void Time::Time_IN()
{cin>>hour>>minute>>second>>sep;
if(hour>0 && minute>0 && second>0){hour=(hour>23)?hour%24:hour;
minute=(minute>59)?minute%60:minute;
second=(second>59)?second%60:second;
sep=(sep!=': ')?': ':sep;}else {hour=minute=second=0; sep=': '; }
} //end of Time_IN

void Time::Time_OUT()
{   cout << setfill( '0' ) << setw( 2 ) << hour << sep << setw( 2 ) << minute << sep
    << setw( 2 ) << second; } //end of Time_OUT

Time Time::add(Time T1,Time T2)
{int t=0;      t=T1.second+T2.second;      second=t>59?t%60:t;
t=(t/60)+T1.minute+T2.minute;      minute=t>59?t%60:t;
t=(t/60)+T1.hour+T2.hour;      hour=t>23?t%24:t;      return *this; } //end of add
```

Introduction

```
Time Time::sub(Time T1,Time T2)
    {if (T1.second>=T2.second) second=T1.second-T2.second;
      else { second=(T1.second+60)-T2.second; T1.minute--; }

    if (T1.minute>=T2.minute) minute=T1.minute-T2.minute;
    else { minute=(T1.minute+60)-T2.minute;T1.hour--; }
    if (T1.hour >=T2.hour) hour= T1.hour-T2.hour;
    else hour=( T1.hour+24)-T2. hour;
    return *this;
} //end of sub

bool Time::is_greater(Time T)
    { if (hour>T.hour) return 1;
      if (hour<T.hour) return 0;
      if (minute>T.minute) return 1;
      else if (minute<T.minute) return 0;
      if(second>=T.second) return 1;
      else return 0;
    }
```

Introduction

```
int main() { Time t1,t2,t3;
    cout<<"t1 = "; t1.Time_OUT(); cout<<endl;
    cout<<"t2 = "; t2.Time_OUT(); cout<<endl;
    t1.Time_IN(); t2.Time_IN();
    t3=t3.add(t1,t2);
    t1.Time_OUT(); cout<<" + "; t2.Time_OUT(); cout<<"="; t3.Time_OUT();cout<<endl;
    if (t1.is_greater(t2)) t3=t3.sub(t1,t2);
    else t3=t3.sub(t2,t1);
    t1.Time_OUT();cout<<" - ";t2.Time_OUT(); cout<<"="; t3.Time_OUT(); cout<<endl;
    system("pause"); return 0; } // end main
```

t1 = 00:00:00

t2 = 00:00:00

10 20 30 :

15 25 35 :

10:20:30 + 15:25:35=01:46:05

10:20:30 - 15:25:35=05:05:05

Press any key to continue . . .

Introduction

- إلا أننا شعرنا ببعض الارتباك والانزعاج لأن الأسلوب الذي اتبعناه يختلف عن الأسلوب الذي اعتدنا استخدامه مع الأنماط مسبقة التعريف في اللغة.
- يلاحظ الفرق بين الأسلوبين من خلال الجدول التالي:

الأسلوب المستخدم	الأسلوب المؤلف	العملية
<code>t1.Time_IN();</code>	<code>cin>>t1;</code>	إدخال t1
<code>t1.Time_OUT();</code>	<code>cout<<t1;</code>	إخراج t1
<code>t3=t3.add(t1,t2)</code>	<code>t3=t1+t2;</code>	جمع t1 و t2
<code>t3=t3.sub(t1,t2)</code>	<code>t3=t1-t2;</code>	طرح t2 من t1
<code>t1.is_greater(t2)</code>	<code>t1>t2</code>	مقارنة t1 مع t2

- لا شك بأنه من المريح التمكن من القيام بإجراء العمليات باستخدام الأسلوب المؤلف.
- تتيح لغة C++ القيام بذلك من خلال إعادة تعريف توابع المعاملات لنتمكن من التعامل مع الأنماط التي نقوم بتعريفها وهذا ما يسمى التحميل الزائد للمعاملات `operator overloading`.

Operator overloading, concept and implementation



1-2 – التحميل الزائد للمعاملات، مفهومه وتحقيقه 1

- تسمح C++ بإضافة عمل على عمليه وتطبيقه على المعطيات الأوليه والمعرفه من قبل المستخدم.
- إن التحميل الزائد للمعاملات: هو كتابة التابع الذي يقوم بالمهمة الجديدة (مؤلف من رأس التابع وجسمه) يختلف هذا التابع عن غيره في أن اسمه يتألف من الكلمة المفتاحية operator متبوعة بإشارة العملية المطلوب تحميلها بشكل زائد.
- يمكن بإجراء التحميل الزائد للمعاملات تحقيق أمرين:
 - ✓ الأول: توسيع استخدام هذه المعاملات بحيث يشمل التعامل مع أنماط جديدة.
 - ✓ الثاني: قيام المعامل بتحقيق عمليات إضافية غير العمليات التي يقوم بها عادة (إلا أن هذا الأمر نادر الاستخدام ولا ينصح باستخدامه لما قد يسببه من التباس في ذهن المبرمج أثناء الاستدعاء لذا سنركز على تحقيق الأمر الأول).
- يتضمن تعريف لغة C++ أساساً بعض المعاملات المحملة تحميلاً زائداً والتي يمكن أن تنفذ أكثر من عمل وذلك حسب نوع العامل على طرفي المعامل (العملية) وبالتالي إن العمليات مرتبطة ومقادة بالأنماط، فعلى سبيل المثال، المعاملان << و >> محملان بشكل زائد حيث يستخدم المعامل >> من أجل القيام بأحد العمليتين التاليتين:

Operator overloading, concept and implementation



1-2 – التحميل الزائد للمعاملات، مفهومه وتحقيقه 2

- ✓ الإدخال ضمن إحدى مجاري الدخل (ومن ثم إلى الذاكرة).
- ✓ الإزاحة بمقدار خانة إلى اليمين للأعداد الثنائية (أي التقسيم على اثنين).
- وبالمثل يستخدم المعامل << من أجل القيام بأحد العمليتين التاليتين:
 - ✓ الإخراج من أحد مجاري الخرج.
 - ✓ الإزاحة بمقدار خانة إلى اليسار للأعداد الثنائية ووضع صفري في الخانة الجديده (أي الضرب باثنين).
- التحميل الزائد للمعاملات يمكن القيام بعملين مختلفين بحسب طريقة الاستدعاء. وكذلك فإن المعاملات الحسابية ومعاملات الإسناد الحسابي وغيرها من المعاملات محملة تحميلاً زائداً، فمثلاً يمكن استخدام المعامل + لإجراء عملية الجمع لقيمتين سواء كانتا من النوع int أو double و اتحاد شريطين محرفيين.
- لتطبيق عملية على أغراض صنف يجب أن تكون تلك العملية محملة تحميلاً زائداً. إلا أنه يوجد لهذه القاعدة استثناءان وهما:
 - ✓ **معامل الإسناد (=) الذي** يمكن استخدامه مع الأغراض بدون أي تعريف ظاهري لتحميله بشل زائد، حيث أنه يقوم بشكل افتراضي بنسخ المعطيات الأعضاء من غرض إلى غرض آخر مماثل، ولكن لهذه العملية محاذير وخطورة عندما يتضمن الغرض أعضاء تؤثر على مناطق محجوزة ديناميكياً في الذاكرة. ويتم تجاوز المشكلة المذكورة من خلال ذكر تعريف واضح لطريقة التحميل الزائد لمعامل الإسناد على هذا النوع من الأغراض.
 - ✓ **معامل العنونة أو المعامل المرجع (&)** الذي يمكن استخدامه مع أغراض أي صنف بدون أي تحميل زائد له إذ يعيد عنوان الغرض ضمن الذاكرة، ولكن أيضاً يمكن تحميل العنونة بشكل زائد.

Operator overloading, concept and implementation



1-2 – التحميل الزائد للمعاملات، مفهومه وتحقيقه 3

• يمكن تحقيق التحميل الزائد للمعامل من خلال كتابة تابع خاص يمكن أن يكون:

- ✓ تابعاً عضواً ضمن الصنف.
- ✓ تابعاً صديقاً للصنف.
- ✓ تابعاً مستقلاً عن الصنف.

• سنقل أثناء دراستنا لموضوع التحميل الزائد للمعاملات الحالة الثالثة، لأن التوابع المستقلة عن الصنف لا يمكن لها أن تصل إلى البيانات الأعضاء الخاصة للصنف،

• وباعتبار أن هندسة البرمجيات تنصح بأن يتم تعريف جميع البيانات الأعضاء للصنف ضمن القسم `private` وبالتالي فإن تعريف توابع مستقلة عن الصنف للتحميل الزائد للمعاملات سيتطلب تعريف توابع وصول إلى البيانات الأعضاء الخاصة ضمن القسم `public`.

Restrictions on operator overloading

تعطي لغة C++ إمكانية لتحميل معظم المعاملات (هناك معاملات لا يمكن تحميلها) بحيث تغدو مرتبطة بنمط المعطيات التي تطبق عليه ويقوم المترجم بتوليد التعليمات المناسبة لطريقة تطبيق العملية واستخدامها. يبين الجدول التالي قائمة بالمعاملات التي يمكن تحميلها والتي لا يمكن تحميلها:

المعاملات التي يمكن تحميلها بشكل زائد								
		&	^	÷	/	*	-	+
*=	-=	+=	<	>	=	!	~	
>>=	<<	>>	=	&=	^=	÷=	/=	
++		&&	<=	>=	!=	==	<<=	
new[]	delete	new	()	[]	,	->	--	
						*->	delete[]	
المعاملات التي لا يمكن تحميلها بشكل زائد								
.		.*	::		?:			

ملاحظة: في بحثنا هذا ناقشنا أمثلة تتضمن بعض المعاملات حيث تناولنا المعاملات الأكثر شيوعاً يمكن لك اختبار التحميل الزائد للمعاملات الباقية في أمثلك الخاصة.

هذا بالإضافة إلى بعض القيود الأخرى:

Restrictions on operator overloading



2-2 – القيود على التحميل الزائد للمعاملات 2

- بعض القيود الأخرى:
- لا يمكن إنشاء عمليات جديدة وهو مبرر للتحميل الزائد.
- لا تتغير أولوية العملية أو أفضلية تنفيذها.
- لا تتغير طريقة تجميع العمليات المحملة بشكل زائد.
- لا يمكن تغيير عدد المعاملات التي تتقبلها عملية ما وتظل العوامل الأحادية أحادية، والعوامل الثنائية ثنائية.
- كل من عوامل التشغيل & و+ و- لها إصدارات أحادية وثنائية.
- لا تتغير طريقة عمل العملية مع الأغراض ذات الأنماط المعرفة مسبقاً لدى تحميلها بشكل زائد.

تلخيص: يتم تحميل العمليات من خلال التوابع التالية وفق الامكان:

- تابعاً عضواً ضمن الصنف.
- تابعاً صديقاً للصنف.
- تابعاً مستقلاً عن الصنف.

Restrictions on operator overloading



2-2 – القيود على التحميل الزائد للمعاملات 3

التابع العضو:

- إذا كان الكائن على يسار العملية ليس من نمط الكائن على اليمين لا يمكن تحميله من خلال تابع عضو بل صديق أو مستقل، وإن كان من نفس النمط يمكن تحميله من خلال تابع عضو إضافة للصديق والمستقل،
- عند تحميله من خلال تابع عضو يتم أخذ الكائن الموجود على اليسار من قبل المؤشر الذاتي `this` والكائن على اليمين يؤخذ بشكل ظاهري من خلال بارامترات التابع إذا كانت العملية تتطلب كائنين مثل `*`، `+`، `-`، `\`، `...`، وفي حال تطلبها كائن وحيد مثل `++` و `--` يتم أخذه من خلال المؤشر.
- يراعى ربط التابع مع الصنف المتضمن لتابع التحميل الزائد عند كتابته خارج الصنف من خلال معامل الارتباط :: وعند عودة الكائن من خلال المؤشر `this` فإن الكائن على اليسار سيتم تغيير قيمته وللمحافظة عليه يعرف كائن في بداية تابع التحميل والعمل عليه وإعادته.

التابع الصديق:

- هو تابع غير عضوي الصنف، بالتالي يسبق بالتعريف صديق `friend`، لا يملك المؤشر الذاتي `this` ويتطلب أخذ الكائنات بشكل ظاهري من خلال بارامترات التابع كائنين للعمليات الثنائية وكائن للعمليات الاحادية، تراعى كتابته خارج الصنف لعدم ارتباطه بالصنف نظراً لعدم احتكار الصداقة من قبل الصنف. ويعمل على كائن مستقل ولا يتغير قيم الكائن على يسار العملية.

التابع المستقل:

- لا يستطيع الوصول إلى الاعضاء الخاصة ضمن الصنف وهناك حلان إما وضع المعطيات ضمن الجزء العام (هذا لا ينصح به) أو كتابة توابع وصول ضمن الجزء العام تصل للمعطيات الخاصة وتنادى من خلال تابع التحميل الزائد.

يأخذ التابع العضو **الشكل العام** التالي:

```
ret-type class-name::operator#(arg-list)
{
    // operations
}
```

حيث أن:

- ✓ ret-type تمثل نوع القيمة المعادة، غالباً ما تكون غرضاً من الصنف الذي يؤثر عليه إلا أنها يمكن أن تكون من أي نوع بيانات صالح.
- ✓ الإشارة # تعبر عن المعامل المراد تحميله بشكل زائد، فعند إنشاء تابع تحميل زائد لمعامل يتم استبدال هذه الإشارة بالمعامل المقصود.
- ✓ arg-list تعبر عن قائمة بارامترات التابع، وتكون فارغة عند تحميل معامل أحادي unary operator، بينما لدى تحميل معامل ثنائي binary operator فإنها ستحتوي على بارامتر وحيد.

- كمثال لاستخدام التحميل الزائد للمعاملات بواسطة توابع أعضاء، نبين فيما يلي نسخة معدلة من المثال المطروح في الفقرة الأولى لإنجاز عمليات الجمع، الطرح والمقارنة لقيم من النمط Time:

Operator overloading using methods as Class Members



3-2 – التحميل الزائد للمعاملات
باستخدام التوابع الأعضاء 2

```
#include <iostream>
#include <iomanip>
using namespace std;
class Time {
public:    Time(int ,int ,int ,char) ;    //constructor
        Time operator +(Time) ;        //finding sum of Time objects
        Time operator -(Time);        //finding sub of time objects
        bool operator >(Time);        //finding greater
        void printUniversal();        //printing Time

private:    int hour;                // 0 - 23 (24-hour clock format)
           int minute;                // 0 - 59
           int second;                // 0 - 59
           char sep;                //:
}; // end clas Time
```

Operator overloading using methods as Class Members



3-2 – التحميل الزائد للمعاملات
باستخدام التوابع الأعضاء 3

```
Time::Time(int hr,int mn,int sc,char sp)
{ hour=(hr>23)?hr/24:hr; minute=(mn>59)?mn/60:mn;
second=(sc>59)?sc/60:sc; sep=(sp==':')?sp:'.';} // end Time constructor
```

```
Time Time::operator +(Time T)
{ int t=0; t=second+T.second; second=t>59?t%60:t;
t=(t/60)+minute+T.minute; minute=t>59?t%60:t;
t=(t/60)+hour+T.hour; hour=t>23?t%24:t; return *this;}// end operator +
```

```
Time Time::operator -(Time T)
{ if (second>=T.second) second=second-T.second;
else { second=(second+60)-T.second; minute--; }
if (minute>=T.minute) minute=minute-T.minute;
else {minute=(minute+60)-T.minute; hour--; }
if(hour >=T.hour) hour=hour - T.hour;
else hour=( hour+24)-T. hour; return *this; }// end operator -
```

Operator overloading using methods as Class Members



3-2 – التحميل الزائد للمعاملات
باستخدام التوابع الأعضاء 4

```
bool Time::operator >(Time T)
{
    if (hour>T.hour) return 1;           else if (hour<T.hour) return 0;
    if (minute>T.minute) return 1;       if (minute<T.minute) return 0;
    if (second> T.second) return 1;       else return 0; }// end operator >
    // in this Function, ELSE it is not necessary because return terminates the function
void Time::printUniversal()
{ cout << setfill( '0') << setw( 2) << hour << sep<<setw( 2) << minute << sep
    << setw( 2) << second<<endl;}//end of printUniversal
int main()
{
    Time t1(10,20,30, ':'); Time t2(15,25,35, ':'); Time t3(0,0,0, ':');
    cout<<"t1 in the initial state = "; t1.printUniversal();
    cout<<"t2 in the initial state = "; t2.printUniversal();
    cout<<"t3 in the initial state = "; t3.printUniversal();
    t3=t1+t2;           //calling the overloaded +
    cout<<"\n\nt1 after calling the + operator = "; t1.printUniversal();
    cout<<"t2 after calling the + operator = ";    t2.printUniversal();
    cout<<"t3 after calling the + operator = ";    t3.printUniversal();
```

Operator overloading using methods as Class Members

```
if (t1>t2) t3=t1-t2;           //calling the overloaded -  
else t3=t2-t1;              //calling the overloaded -  
cout<<"\n\n t1 after calling the - operator = "; t1.printUniversal();  
    cout<<"t2 after calling the - operator = "; t2.printUniversal();  
cout<<"t3 after calling the - operator = "; t3.printUniversal();  
system ("pause"); return 0;} // end main
```

الخرج:

```
t1 in the initial state = 10:20:30  
t2 in the initial state = 15:25:35  
t3 in the initial state = 00:00:00
```

```
t1 after calling the + operator = 01:46:05  
t2 after calling the + operator = 15:25:35  
t3 after calling the + operator = 01:46:05
```

```
t1 after calling the - operator = 01:46:05  
t2 after calling the - operator = 13:39:30  
t3 after calling the - operator = 13:39:30  
Press any key to continue . . .
```

- بمناقشة البرنامج السابق ونتائج تنفيذه، يمكن أن نسجل الملاحظات التالية:
 - ✓ احتوى التابع `(+) operator` على بارامتر واحد على الرغم من أنه يقوم بالتحميل الزائد للمعامل الثنائي `+`. السبب وراء ذلك هو أن الوسيط على يسار المعامل `+` يتم تمريره إلى التابع بشكل ضمني من خلال المؤشر `this`، أما الوسيط إلى يمين المعامل فيتم تمريره من خلال البارامتر `T`.
 - ✓ إن حقيقة كون الوسيط على يسار المعامل يتم تمريرها من خلال المؤشر `this` تعني أمراً مهماً، وهو إنه عند التحميل الزائد للمعاملات الثنائية، فإن الغرض الموجود على يسار المعامل هو الذي يولد الاستدعاء لتابع المعامل.
 - ✓ من الشائع أن يقوم تابع المعامل المحمل بشكل زائد بإعادة غرض من الصنف الذي يؤثر عليه، مما يتيح للمعامل أن يستخدم في تعابير أكبر، مثلاً. إذا أعاد التابع `(+) operator` غرضاً من نوع آخر، فإن التعبير التالي لن يعود صالحاً: `(t3 = t1 + t2;)`. فمن أجل أن يتم إسناد ناتج جمع الغرضين `t1` و `t2` إلى الغرض `t3` فناتج هذه العملية يجب أن يكون غرضاً من النوع `Time`.
 - ✓ إن كون `(+) operator` بعيد غرضاً من النوع `Time` يجعل من الممكن استخدام تعابير من النوع: `((t1+t2).printUniversal();)`. في هذه الحالة تقوم العملية `t1+t2` بإضافة `t2` إلى `t1` وتوليد غرض مؤقت ووضع الناتج فيه وينتهي وجوده بمجرد انتهاء استدعاء التابع `printUniversal()`.
 - ✓ قام تابع المعامل `+` وتابع المعامل `-` بتغيير قيمة البارامتر المستدعي (الغرض الموجود على الطرف الأيسر من العملية) والسبب في ذلك كتابة تابع المعامل بحيث يستخدم المؤشر `this`.
 - ✓ كان بالإمكان جعل التابع لا يؤثر في قيم الوسيط اليساري الممرر ضمناً وهو الاستخدام التقليدي للمعاملين `+` و `-`، يكون ذلك بتعديل تابعي المعاملين بالشكل التالي:

Operator overloading using methods as Class Members

```
Time Time::operator +(Time T)
{Time temp(0,0,0, ':'); int t=0;
t=second+T.second; temp.second=t>59?t%60:t;
t=(t/60)+minute+T.minute; temp.minute=t>59?t%60:t;
t=(t/60)+hour+T.hour; temp.hour=t>23?t%24:t; return temp; }
```

```
Time Time::operator -(Time T)
{Time temp(0,0,0, ':');
if (second>=T.second) temp.second=second-T.second;
else { temp.second=(second+60)-T.second; minute--; }
if (minute>=T.minute) temp.minute=minute-T.minute;
else { temp.minute=(minute+60)-T.minute; hour--; }
if(hour >=T.hour) temp. hour =hour-T.hour;
else temp. hour =(hour+24)-T. hour;
return temp;
}
```

Operator overloading using methods as Class Members



3-2 – التحميل الزائد للمعاملات باستخدام التوابع الأعضاء 8

سيكون الخرج:

```
t1 in the initial state = 10:20:30  
t2 in the initial state = 15:25:35  
t3 in the initial state = 00:00:00
```

```
t1 after calling the + operator = 10:20:30  
t2 after calling the + operator = 15:25:35  
t3 after calling the + operator = 01:46:05
```

```
t1 after calling the - operator = 10:20:30  
t2 after calling the - operator = 15:25:35  
t3 after calling the - operator = 05:05:05  
Press any key to continue . . .
```

Operator overloading using methods as friend methods

3-2 – التحميل الزائد للمعاملات باستخدام التوابع الصديقه 1

- يمكن أن يتم التحميل الزائد للمعاملات من أجل التعامل مع صنف معين باستخدام تابع غير عضو، يكون هذا التابع عادة صديقاً للصنف (نذكر أنه يمكن للتابع غير العضو أن يكون مستقلاً أي غير صديق إلا أن ذلك سيتطلب تحقق أحد أمرين، إما أن البيانات الأعضاء للصنف معرفة ضمن القسم public كي يكون هناك إمكانية للوصول إليها باستخدام تابع غير عضو وإما أن يحتوي الصنف على توابع وصول معرفة ضمن القسم public إلى البيانات الأعضاء الخاصة مثل set و get).
- بما أن التابع الصديق ليس تابعاً عضواً في الصنف، فإنه لا يستطيع استخدام المؤشر this وبالتالي فإن تابع المعامل المحمل بشكل زائد يتم تمرير الوسطاء إليه بشكل صريح ضمن لائحة البارامترات، وهذا يعني أن التابع الصديق الذي يقوم بالتحميل الزائد لمعامل ثنائي يملك بارامترين، والتابع الصديق الذي يقوم بالتحميل الزائد لمعامل أحادي يملك بارامتراً واحداً.
- فيما يلي نسخة معدلة من البرنامج السابق الذي يقوم بالتحميل الزائد لمعاملات الجمع، الطرح والمقارنة. حيث تم استخدام توابع صديقه لإنجاز المطلوب:

```
#include <iostream>
```

```
#include <iomanip>
```

```
using namespace std;
```

Operator overloading using methods as friend methods



3-2 – التحميل الزائد للمعاملات
باستخدام التوابع الصديقه 2

```
class Time {
    friend Time operator +(Time,Time); //finding sum
    friend Time operator -(Time,Time); //finding sub
    friend bool operator >(Time,Time); //finding greater
public:
    Time(int ,int ,int ,char) ; //constructor
    void printUniversal(); //printing Time
private:
    int hour; // 0 - 23 (24-hour clock format)
    int minute; // 0 - 59
    int second; // 0 - 59
    char sep; //:
}; // end clas Time
Time::Time(int hr,int mn,int sc,char sp)
{ hour=(hr>23)?hr/24:hr; minute=(mn>59)?mn/60:mn;
second=(sc>59)?sc/60:sc; sep=(sp==':')?sp:'.'; } // end Time constructor
```

Operator overloading using methods as friend methods



3-2 – التحميل الزائد للمعاملات
باستخدام التوابع الصديقه 3

```
Time operator +(Time T1,Time T2)
{ Time temp(0,0,0, ':');      int t=0;
  t=T1.second+T2.second; temp.second=t>59?t%60:t;
  t=(t/60)+T1.minute+T2.minute; temp.minute=t>59?t%60:t;
  t=(t/60)+T1.hour+T2.hour; temp.hour=t>23?t%24:t; return temp;
} //end of operator +
```

```
Time operator -(Time T1,Time T2)
{ Time temp(0,0,0, ':');
  if (T1.second>=T2.second) temp.second=T1.second-T2.second;
  else { temp.second=(T1.second+60)-T2.second; T1.minute--; }
  if (T1.minute>=T2.minute) temp.minute=T1.minute-T2.minute;
  else { temp.minute=(T1.minute+60)-T2.minute; T1.hour--; }
  if(T1.hour >=T2.hour) temp.hour =T1.hour - T2.hour;
  else temp.hour =(T1. hour+24)-T2. hour;return temp;
} //end of operator -
```

Operator overloading using methods as friend methods



3-2 – التحميل الزائد للمعاملات
باستخدام التوابع الصديقه 4

```
bool operator >(Time T1,Time T2)
```

```
{   if (T1.hour>T2.hour)   return 1;
    else if (T1.hour<T2.hour)   return 0;
    if (T1.minute>T2.minute)   return 1;
    if (T1.minute<T2.minute)   return 0;
    if(T1.second>T2.second)   return 1;
return 0; }//end of operator >
```

```
void Time::printUniversal()
```

```
{   cout << setfill( '0') << setw( 2) << hour << sep<<setw(2) << minute << sep
    << setw( 2) << second<<endl; } //end of printUniversal
```

```
int main()
```

```
{   Time t1(10,20,30,':'); Time t2(15,25,35,':'); Time t3(0,0,0,':');
    cout<<"t1 in the initial state = ";   t1.printUniversal();
    cout<<"t2 in the initial state = ";   t2.printUniversal();
    cout<<"t3 in the initial state = ";   t3.printUniversal();
```

Operator overloading using methods as friend methods



3-2 – التحميل الزائد للمعاملات
باستخدام التوابع الصديقه 5

```
t3=t1+t2;           //calling the overloaded +
    cout<<"\n\nt1 after calling the + operator = "; t1.printUniversal();
    cout<<"t2 after calling the + operator = ";     t2.printUniversal();
    cout<<"t3 after calling the + operator = ";     t3.printUniversal();

    if (t1>t2)      //calling the overloaded >
        t3=t1-t2;  //calling overloaded -
    else
        t3=t2-t1;  //calling overloaded -

    cout<<"\n\nt1 after calling the - operator = "; t1.printUniversal();
    cout<<"t2 after calling the - operator = ";     t2.printUniversal();
    cout<<"t3 after calling the - operator = ";     t3.printUniversal();
    system ("pause");
return 0;
} // end main
```



انتهت المحاضرة الرابعة