تعلم البرمجة باستخدام لغة
Liberty Basic 4.01

.....

اعداد:gameprogrammer

اذا لم تكن تعرف كيفية استخدام محرر هذه اللغة انتقل الى الصفحة 104 لدراسة كيفية استخدام القوائم الموجودة في محرر هذه اللغة.

هذا الكتاب يختص بتعليم ٩٥% من لغة الـ Liberty Basic ويتطرق للمواضيع التالية:

١.مفردات لغة Basic القديمة المستعملة مع هذه اللغة.
٢.التعامل مع النصوص.
٣.التعامل مع انظمة الملفات.
٤.الرسم باستخدام البرمجة.
٥.برمجة الالعاب بواسطة الـ sprites .
٦.برمجة النوافذ (نوافذ نظام Windows).
٧.التعامل مع الـ API ,مع ملاحظة ان الكتاب لايشرح هذه الدوال و لكن يشرح كيفية التعامل معها لبرمجة النظام Windows بصورة سريعة و سهلة.و يشمل ايضا التعامل مع مكتبات الربط الديناميكية DLLs .
٨.التعامل مع المنافذ من نوع serial ports و hardware Input/Output ports, مع ملاحظة ان هذا الكتاب لايشرح ماهي هذه المنافذ بصورة مفصلة و لكن يشرح فقط كيفية التعامل معها عن طريق هذه اللغة,ويجب عدم تصميم أي برامج تتعامل مع الـports مالم تكن قد درست تكوين هذه الـports ,وكل المعلومات الواردة في هذا الدرس هي معلومات مترجمة من ملف الـhelp و من بعض صفحات الانترنت.

جزء كبير من هذا الكتاب هو ترجمة لملف الـhelp المرفق مع لغة Liberty Basic . اوامر و جمل لغة Basic تم شرحها بصورة مختصرة لأن الكاتب يفترض ان مستخدم هذا الكتاب قد تعلم لغة Basic و هي لغة مناسبة للتعلم في بداية كتابة البرامج.

الكاتب غير مسؤول عن أي ضرر أو عطل يصيب أي برامج أو أي جزء من اجزاء الحاسوب أو أي جهاز ملحق بالحاسوب نتيجة لاستخدام المعلومات الموجودة في هذا الكتاب.

تعريف البرمجة:

البرمجة هي عبارة عن مجموعة من الاوامر التي تدخل الى الحاسوب عن طريق لوحة المفاتيح لتؤدي غرض معين كحل المسائل الرياضية مثلا.و البرمجة تتم عن طريق لغات البرمجة ولغات البرمجة تنقسم الى نوعين:

١.لغات المستوى العالي:مثل لغة Basic,c++,Java,Liberty Basic و لغات اخرى..
٢.لغات المستوى الواطئ:مثل لغة الماكنة Machine Language ولغة التجميع Assembly.

ماذا تحتاج لكتابة البرامج بأي لغة؟

تحتاج الى محرر نصوص تكتب فيه اوامر اللغة التي تريد استخدامها للبرمجة.

ماهو محرر النصوص؟

محرر النصوص هو برنامج كبرنامج Notepad و Microsoft Word تستطيع من خلاله الكتابة و معالجة النصوص.

عندما تكتب برنامج بأي لغة فأن البرنامج يترجم الى لغة الـ Assembly ثم الى لغة الـ Machine Code لتذهب الى معالج الحاسبة ثم الى وحدة الاظهار لاظهار النتائج على الشاشة.

ومحررات النصوص تقسم الى قسمين:

١.Compilers:تقوم بترجمة البرنامج الى لغة الـ Machine Code قبل تنفيذه ,فيكون تنفيذ البرنامج سريع جدا لأنه لايحتاج الى ترجمة.

٢.Interpretters:يقوم بترجمة سطر من البرنامج و تنفيذه ثم ترجمة السطر الاخر و تنفيذه و هكذا ,وفي كل مرة تنفيذ تعاد العملية من جديد.

اللغات التي تستعمل الـ Compilers تكون البرامج المكتوبة ضمنها اسرع من البرامج المكتوبة من خلال الـ Interpretters,ولكن الـ Compilers لاتستطيع تحديد السطر الذي يحتوي خطأ,فيوجه الـ Compiler رسالة تحذيرية عن وجود خطأ في البرنامج لكن لايعلم اين الخطأ بالضبط.أما الـ Interpretters فتترجم كل سطر بصورة مفردة فتستطيع تحديد مكان الخطأ.

أنواع الاخطاء:

لكل لغة قواعد فيجب على المبرمج اتباع هذه التعليمات و الا حصل خطأ في البرنامج أي ان المبرمج يتكلم مع الحاسبة باللغة التي تفهمها عن طريق مترجم (Compiler أو Interpretter),وتقسم الاخطاء الى عدة انواع:

١.اخطاء لغوية Syntax Errors :وهي الاخطاء التي تنتج عن خطأ لغوي في كتابة الاوامر أي نسيان حرف ما مثلا.
٢.اخطاء منطقية:تنتج عن البرمجة و الحاسبة لاتستطيع اكتشافها,مثلا تريد ان تصمم برنامج لتعويض عدد من الاعداد في القانون الرياضي x+1 فربما المبرمج قد اخطأ في كتابة التعبير الرياضي و كتبه على الشكل التالي x-1 فستكون النتيجة خاطئة و الحاسبة لن تكشفها لأنها لاتعرف ماهي المعادلة الاصلية فالحاسبة تتصرف حسب المعادلة المعطاة لها في البرنامج.
معظم لغات البرمجة تحوي محرر نصوص خاص بها تقوم بكتابة البرنامج فيه ثم تنفذ البرنامج عن طريق Run و هو أحد الاوامر الموجهة للحاسوب لتنفيذ البرنامج و تجده غالبا في احدى قوائم محرر النصوص,عند التنفيذ في لغات البرمجة الحديثة تلاحظ ظهور نافذة جديدة لتبين ماذا يحصل اثناء التنفيذ.

مبدأ عمل البرامج:
كل برنامج مهما تعقد يتضمن ثلاثة مراحل هي:

١.الادخال:يتضمن ادخال قيم الى الحاسب لتتم معالجتها أي اجراء العمليات الرياضية عليها.
٢.المعالجة:تتضمن اجراء العمليات على الاعداد التي وصلت الى المعالج عن طريق الذاكرة خلال عملية الادخال.
٣.الاخراج:ترسل البيانات التي انتهى المعالج من معالجتها الى وحدة الاخراج و تشمل وحدة الاظهار و تمثل بالـ Video Card الذي يقوم بترجمتها الى اشارات تفهمها الشاشة ليعرضها على الشاشة.

لتوضيح هذه الافكار نأخذ هذا المثال:
بائع طلب من مبرمج تصميم برنامج ليحسب سعر المواد المباعة حسب عددها و سعرها ,فالمبرمج يجب عليه القيام بالخطوات التالية:

١.ماهو المطلوب من البرنامج؟حساب سعر المواد.
٢.كيف نحسب سعر المواد؟تحسب عن طريق (سعر المادة×عدد المواد).
٣.ماذا سيطلب البرنامج من المستخدم؟سيطلب سعر المادة و عدد المواد المباعة.

ان اغلب المبرمجين يتبعون هذه الطريقة أي معرفة ماهو ناتج البرنامج (الاخراج) و ماهي العملية التي ستجري على البيانات (معالجة) و ماذا يتطلب لحساب الناتج (الادخال).
ستحتاج في البرنامج الى مخازن تخزن فيها اعداد او نصوص معينة لتغير محتويات المخازن أي وقت تريد,هذه المخازن تسمى متغيرات Variables وهي من الاشياء الاساسية في كل لغات البرمجة وهي على نوعين:

١.متغيرات مقطعية String Variables:تستخدم لخزن المقاطع و النصوص بين علامات الاقتباس (") ويمكن أن تحمل أي اسم ويجب ان تبدأ بحرف و تنتهي بالرمز ($) لتبين انها متغيرات مقطعية.
٢.متغيرات عددية:تستخدم لخزن الاعداد لتمثيلها في العمليات الرياضية مثلا,و تبدأ بحرف و لاتنتهي بالرمز $.

تصنف لغة Liberty Basic الى لغات المستوى العالي و هي لاتستخدم Compiler أو Interpretter لكنها تستخدم مايعرف بـ Virtual Machine و Run-Time Engine .

أي ان اوامر هذه اللغة لاتترجم الى لغة الـ Machine Code بل يقوم الـ Run-Time Engine بترجمة البيانات الى لغة يفهمها و عرض النتائج على الشاشة.وعلى العموم ان لغة Liberty Basic سريعة و قوية و سهلة.

يمكن الحصول على هذه اللغة من الموقع www.libertybasic.com
ولكن يجب شراء هذه اللغة عن طريق الانترنت وهي على نوعين Gold Edition و Silver Edition و النوع الاول افضل من الثاني لأنه يحتوي على مميزات اكثر.
أما اذا كنت تريد نسخة مجانية منها يمكن الحصول عليها من الموقع www.justbasic.com
وهي نفس لغة Liberty Basic لكن مع أوامر اقل بكثير من النسخة التجارية.

بعد تشغيل Liberty Basic سيظهر برنامج ترحيبي ليشرح صفات هذه اللغة ,من قائمة File اختر New لازالة هذا البرنامج حاليا.

سنبدأ الان بشرح الجزء المتعلق بلغة Basic القديمة.

١.لايوجد ترقيم للاسطر بهذه اللغة.
٢.لايوجد الامر New .

الجملة print:
تستخدم لطبع البيانات على الشاشة و تستخدم بعدة صيغ (اكتب البرنامج التالي في محرر اللغة):

```
Print “this is my first program"
Print “line 1”
Print “line 2”;
Print “line 3”
Print “line 4”
Print “line5”,:print "line 6”
```

الجملة print تقوم بطبع ماموجود بين علامتي الاقتباس (").
الجملة الاولى طبعت على السطر الاول و الثانية على السطر الثاني أي ان كل جملة print تنقلنا سطرا الى الاسفل.لكن الجملة الثالثة و الرابعة طبعت على نفس السطر لوجود الرمز (;) بعد جملة print الثالثة,أي ان وجود هذا الرمز بعد جملة print يؤدي الى جعل ناتج جملة print التي بعدها على نفس السطر و بلا فاصل بين الجملتين,يمكن ان تحصل على الرمز (;) بالضغط على المفتاح الذي يحمل الحرف (ك) لكن بعد التحويل الى اللغة الانكليزية.و الجملة الخامسة لم تظهر معها على نفس السطر السابق لأن الجملة الرابعة لاتحوي على هذا الرمز لذلك سيؤدي الى الانتقال الى سطر جديد.الجملة السادسة يمكن اعتبارها جملتين أي انها تعتبر عبارتين عندما تفصل بين جملتين على نفس السطر بالرمز (:) وهما تظهران على نفس السطر لكن بمسافة معينة نتيجة لاستخدام رمز الفارزة (,) الذي له نفس استعمال الـ(;) لكن مع ترك مسافة اكبر.
يمكن استخدام print لحساب العمليات الرياضية:

```
print 5+6
print 20*20
print 20/10
print 20-10
print 2^4+2-1
print (2+4)*4+(6+4)
print "product is";2*4
x=4
```

```
y=x*2
print "y=";y
```

انتبه الى ان اننا لم نستخدم علامات الاقتباس لأننا نريد طبع قيمة رياضية,الجمل الاربعة واضحة أما الجملة الخامسة فالرمز (^) هو الرفع للقوة 4^2 أي (٢*٢*٢*٢), و الاسبقية في تنفيذ العمليات هي للتعابير بين الاقواس ثم الرفع للقوة ثم الضرب و القسمة و الجمع و الطرح.الجملة السابعة سيطبع 4*2 أي 8 الى جانب العبارة بين علامتي الاقتباس.و الجملة الثامنة و التاسعة فهي مثال على المتغيرات العددية x=4 أي ان انك تطلب من الحاسبة ان تجعل قيمة x تساوي 4 ففي هذه الحالة قيمة y ستكون x*2 أي 8 و في الجملة العاشرة ستطبع النتيجة مع جملة تبين ان هذه قيمة المتغير y.
نفذ البرنامج التالي:

```
v1$=”liberty”
v2$=”basic”
print “the best language is”;” “;v1$;” “;v2$
```

هنا استخدمنا المتغيرات المقطعية حيث هذا مثال على دمج الكلمات في جملة واحدة,الجملتين الاولى و الثانية هي تعويض للكلمتين في متغيرين لاستخدام اسهل للكلمات,الجملة الثالثة تحوي على علامتي اقتباس بينهما مسافة فارغة لكي تضع هذه المسافة لتفصل بين الكلمات حيث المسافة الفارغة تعتبر كلمة ايضا.ويتم الفصل بين الكلمات المراد كتابتها على نفس السطر بالرمز (;).أعد كتابة البرنامج بالشكل التالي:

```
v1$=”liberty”
v2$=”basic”
print “the best language is”; v1$; v2$
```

ستلاحظ اختفاء الفراغات أو المسافات بين الكلمات.
نفذ البرنامج التالي:

```
x=4
x=7
print "the sum of”;” “;x;” “;”and”;” “;”3”;” “;”is”;” “;x+3
```

ستكون النتيجة التي تظهر على الشاشة تساوي 10 لأن x=7 وسبب ذلك ان كل قيمة تعطى للمتغير تحذف القيمة القديمة و تحل محلها فمثلا x كانت تساوي 4 لكن العبارة التي بعدها جعلت x يساوي 7 .
نستنتج ان الرمز (;) يقوم بدمج الجملة او الكلمة التي تأتي بعده مع الجملة او الكلمة التي قبله وهكذا.
نفذ البرنامج التالي:

```
print “hello”,”world”
```

سيكون الناتج كلمتي hello و world بينها عدة مسافات نتيجة استخدام الفارزة.

ملاحظة :عند تنفيذ البرنامج تلاحظ ظهور نافذة تقوم بعرض ناتج البرنامج ,هذه النافذة تسمى Main Window وتسمى للاختصار mainwin .

الجملة input:

تستخدم لاستقبال البيانات من المستخدم لغرض معالجتها و تشمل بيانات عددية او مقطعية وتخزن قيمة البيانات المدخلة بواسطة المستخدم في المتغير الموجود مع هذه الجملة.
نفذ البرنامج التالي:

```
print "What is your name"
input n$
print "How old are you"
input h
print "You are";" ";n$;" ";"and your age is";" ";h;" ";"years old.."
```

لاحظ الجملة الثانية ينتج عنها عند التنفيذ علامة استفهام ليطلب منك ادخال أي قيمة مقطعية لكن لن تعرف انه يجب ادخال قيمة مقطعية الا اذا دلت جملة ما على ذلك و لهذا قمنا بكتابة print ضمن البرنامج,أي ان input يستقبل البيانات المدخلة و يخزنها في المتغير الموجود ضمنه و هو في هذه الحالة $n و في الجملة الرابعة هو h و هو متغير عددي لأن المطلوب ادخال العمر و هو عدد.
يوجد صيغة اخرى لـ input تغنيك عن print وهي في البرنامج التالي:

```
input "number of computers ";nc
input "price of one computer ";pc
total=nc*pc
print "the total price of";" ";nc;" ";"computers is";" ";total
```

يبدو السطر الرابع معقد لنعيد كتابته بالشكل التالي:

```
print "the total price of ";nc;" computers is ";total
```

لاحظ انه تم الاستغناء عن المساحات الفارغة (" ") وتعويضها بمساحات فارغة ضمن الجملة المطبوعة أي عند كتابة شئ ما داخل علامات الاقتباس اترك فراغا واحدا ثم ابدأ بالكتابة عندما تريد ان تكون النتيجة مرتبة.
لاحظ الصيغة الاخرى لـ input هي صيغة تحوي على جملة تريد طباعتها للتوضيح عن نوع القيم المراد ادخالها, و متغير يخزن القيمة المدخلة.
ملاحظة :في البرامج الكبيرة ينصح ان تكون اسماء المتغيرات تدل على محتواها مثلا السعر price العدد number أو num وغيرها..

الجملة cls:
وهي اختصار لـ clear screen أي مسح الشاشة,هذا الامر يستخدم لمسح محتويات نافذة main window .وتستعمل هذه الجملة بمجرد كتابتها cls فقط.

الجملة goto:
واضح من معناها انها تستخدم لتوجيه البرنامج الى سطر معين,ولكن السؤال هو كيف نستطيع ان نخبر البرنامج الى أي سطر يجب ان يتوجه,الجواب هو باستخدام الـ labels كما في المثال التالي:

```
cls
print "hello,this program can add two numbers"
[addition]
input "your first number ";n1
```

```
input "your second number ";n2
print n1;"+";n2;"=";n1+n2
print
goto [addition]
```

الـ labels هو العناوين أي تسمية قسم من البرنامج لكي نستطيع التوجه اليه بجملة goto مثلا كلما اردنا ,وذلك بكتابة اسم العنوان بالقرب من goto ويجب وضع الرمزين ([]) و ([) واسم العنوان يكون بينهما.في البرنامج السابق سيتم التوجه الى العنوان [addition] كلما انتهى البرنامج.و سبب استخدام جملة print بدون كتابة أي عبارة بجانبها هو لترك سطر فارغ.

البرامج الفرعية:

تستخدم البرامج الفرعية في نهاية البرنامج الرئيسي وذلك لتكرار تأدية مهمات معينة وللبرنامج الفرعي عنوان معين label ويتم التوجه الى البرنامج الفرعي باستخدام جملة gosub ويأتي بعدها الـ label المراد التوجه اليه,ثم بعد تأدية المهمة الموجودة في البرنامج الفرعي يتم العودة الى السطر الذي يلي السطر الذي تفرع منه البرنامج الى البرنامج الفرعي باستخدام جملة return .

نفذ البرنامج التالي:

لاينصح باستخدام البرامج الفرعية في البرامج الصغيرة لكن تظهر اهميتها في البرامج الكبيرة و البرنامج التالي برنامج صغير لكن يستعمل لتوضيح استخدام البرامج الفرعية.

```
Cls
Gosub [drawrectangle]
Gosub [drawline]
Print "the drawing finished.."
Wait

[drawrectangle]
print "************************"
print "*                      *"
print "*                      *"
print "************************"
return

[drawline]
print "************************"
return
```

سيتفرع البرنامج لينفذ الاوامر الموجودة في العنوان [drawrectangle] ثم سيعود عندما يجد return الى الجملة الثالثة (الجملة التي تفرع منها هي الثانية لذلك سيعود الى الجملة الثالثة) ثم يتفرع الى العنوان [drawline] ثم يرجع و يطبع الجملة the drawing finished ثم يجد wait فينتظر و لايتجاوز هذه العبارة أي ان هذه العبارة تجعل البرنامج في حالة انتظار لكي لا يتجاوزها الى الاوامر الباقية.

الجمل الشرطية:

هي جمل تقوم باختبار بعض القيم و التوجه الى عنوان ما او عمل حسابات معينة عندما يتحقق الشرط.يتم هذا عن طريقة عبارة if/then وهي على عدة صيغ.

```
If condition then result
```

If condition then result1 else result2

```
If condtion then
Result1a
Result2a
. . .
. . .
end if

If condtion then
Result1a
Result2a
. . .
. . .
else
result1b
result2b
. . .
. . .
end if
```

الـ condition هو شرط و غالبا مايكون مقارنة ما بين قيمتين,و الـ result هي النتيجة و هي تكون اوامر نريد تنفيذها عند تحقق الشرط condition .

نفذ البرنامج التالي:

```
print "enter two numbers.."
input "first number ";n1
input "second number ";n2
if n1>n2 then print n1;" is greater"
if n1<n2 then print n2;" is greater"
if n1=n2 then print "they are equal"
```

وهذا مقطع من برنامج لعبة كمبيوتر و هو يستخدم للتأكد من وجود وقود في الطائرة و التصرف تبعا لذلك..لاينفذ هذا البرنامج بصورة صحيحة لأنه مقطع من برنامج..

```
if fuel=0 then
print "there is no fuel in the plane"
gosub [lose]
else
if damage=0 then gosub [lose]
print "there is fuel in the plane"
end if
```

لاحظ انه يمكن استخدام جملة if داخل جملة if اخرى و هكذا..

الجملة end if توضع في نهاية جملة if الممتدة على عدة اسطر..

وهناك جملة اخرى في الشروط هي select case وتستعمل لاختبار قيمة متغير او تعبير رياضي.

نفذ البرنامج التالي:

```
input a
select case a

case 1
print "it is one"

case 2
print "it is two"

case else
print "I do not know"

end select
```

هنا المتغير المراد اختباره هو المتغير a و الرقم المعطى بجانب كل كلمة case يمثل قيمة المتغير ,فاذا كانت قيمته ٢ سيهمل case 1 ويطبع it is two ,ويمكن ترجمة هذا البرنامج باستخدام الـ if\then

```
input a
if a=1 then print "it is one"
if a=2 then print "it is two"
if a<>2 and a<>1 then print "I do not know"
```

ويمكن ان تختبر قيمتين لمتغير واحد

```
select case a

case 1,2
print "it is one or two"

case 3,4,5,8,9
'some code..
end select
```

أي عندما تكون قيمة a تساوي ١ او ٢ يتصرف تبعا للـ case 1,2 و عندما تكون قيمته 3,4,5,8,9 يتصرف تبعا لذلك.

ويمكن اختبار ناتج تعبير رياضي

```
select case a+3
case 4
'some code
case 6
'some code
end select
```

الرمز (') يستعمل في أي مكان لاضافة توضيح و لايدخل ضمن ترجمة البرنامج بل قد يستعمله المبرمج للتوضيح للمبرمجين الاخرين عن الغرض من البرنامج.ويمكن عدم كتابة اسم المتغير او التعبير الرياضي وكتابة مقارنة في جمل case

```
select case

case (a>10)
'some code
case (a<10) or (a=10)
'some code
```

```
case else
'some code
end select
```

اما عمليات المقارنة هي:

A=B A equals B
A>B A greater than B
A<B A less than B
A<>B A not equal B
A<=B A less than or equal B
A>=B A greater than or equal B

واذا اردنا تحقق شرطين معا نضع and بين الشرطين و اذا اردنا تحقق احدهما نضع or
واذا اردنا تحقيق عكس الشرط نضع not .

في لغات البرمجة القيمة الصحيحة تكون قيمتها ١ و القيمة الخاطئة تكون قيمتها ٠ ,

```
if (a>b) and (a>c) then result
if (a>b) or (a=b) then result
if not(a>b) then result
```

truth table for AND

x	y	returned value
…	…	……
0	0	0
1	0	0
0	1	0
1	1	1

truth table for OR

x	y	returned value
…	…	……
0	0	0
1	0	1
0	1	1
1	1	1

truth table for XOR

x	y	returned value
…	…	……
0	0	0
1	0	1
0	1	1
1	1	0

في هذه الحالة تبين ان x هو الشرط الاول و y هو الشرط الثاني و عندما تكون قيمة أي منهما صفر معناه ان الشرط فاشل مثلا قيمة a=4 فعندما تضع الشرط التالي:

```
if (a>4) then result
```

هنا الشرط غير متحقق أي قيمته 0 أي انه لا تتحقق النتيجة بعد then .

```
if (a>4) or (a=4) then result
```

هنا a>4 تعود بالقيمة 0 و a=4 تعود بالقيمة 1 و قيمة 0 OR 1 حسب الجداول هي 1 ,فهنا الشرط يتحقق,أما NOT فهي تقوم بعكس القيمة أي 1 تكون 0 و بالعكس. وتستعمل هذه الاوامر ايضا في الاعداد العشرية بتحويلها الى الاعداد الثنائية ثم اجراء العمليات المنطقية عليها..

print 3 or 4

يقوم بتحويل 3 الى 011 و 4 الى 100

011
100

يعمل الان عملية OR المنطقية بين كل bit و bit مقابل له

……..OR
111

الناتج 7 بالنظام العشري.

By:gameprogrammer

الحلقات التكرارية:
توجد ثلاثة انواع من الجمل التكرارية وهي:

١.for\next
نفذ البرنامج التالي:

```
for g=1 to 10
print g
next g
```

تلاحظ ان ناتج البرنامج هو مجموعة من الاعداد تمتد بين 1 الى 10 أي ان الجمل الموجودة بين جملة for و جملة next تنفذ عشر مرات,ويسلك البرنامج السلوك التالي: تكون قيمة g هي 1 فينفذ البرنامج العبارات القادمة الى ان يصل next التي يجب ان يتبعها اسم المتغير المستعمل في عبارة for و هو هنا g ,ثم يرجع الى for و تزداد قيمة g و ينفذ هذه العبارات و هكذا الى ان يبلغ الرقم 10 .
نفذ البرنامج التالي:

```
for g=1 to 10 step 2
print g
next g
```

الناتج سيكون من 1 الى 10 مع ترك رقم واحد بين كل رقمين نتيجة استخدام step أي خطوة أي المقصود هنا انه حساب من 1 الى 10 مع زيادة بمقدار 2 .

نفذ البرنامج التالي:

```
for g=10 to 1 step –1
print g
next g
```

تستخدم الاعداد السالبة في جملة for عند الحساب من قيمة اكبر الى اخرى اصغر.
للخروج من الحلقة التكرارية for\next دون اكمالها يمكن استخدام الامر EXIT FOR.

٢.while/wend
يقوم البرنامج بتنفيذ الجمل مابين هاتين العبارتين حسب الشرط الذي يأتي بعد while

نفذ البرنامج التالي:

```
while a<9
print a
a=a+1
wend
```

سيقوم البرنامج بطباعة الاعداد بين 1و 8

للخروج من داخل الحلقة التكرارية while يجب ان تكتب exit while

٣.do/loop

```
do while condition
```
يتم تنفيذ الجمل بين هاتين العبارتين مادام الشرط متحققا
```
loop
```

```
do until condition
```
يتم تنفيذ العبارات بين هاتين العبارتين الى ان يتحقق الشرط
```
loop
```

```
do
```
يتم تنفيذ الجمل بين هاتين العبارتين مادام الشرط متحققا
```
loop while condition
```

```
do
```
يتم تنفيذ العبارات بين هاتين العبارتين الى ان يتحقق الشرط
```
loop until condition
```

نفذ البرنامج التالي:

```
print "print a zero"
do
print a
a = a + 1
loop while a > 10
print
print "print 1 to 9"
do
print a
a = a + 1
loop while a < 10
print
print "print a zero"
do
print b
b = b + 1
loop until b = 1
print
print "print 1 to 9"
do
print b
b = b + 1
loop until b = 10
print "print 1 to 3"
a = 1
do while a <= 3
print a
a = a + 1
loop
print
print "print 9 to 7"
b = 9
do until b = 6
print b
b = b - 1
loop
print
print "don't print anything"
do while c = 10
print c
c = c + 1
loop
end
```

ملاحظة :الجملة end تقوم بانهاء البرنامج.

المصفوفات arrays:

١.المصفوفات بالبعد الواحد:

في حالة احتجت الى ٥٠٠ متغير مثلا لحفظ اسم ٥٠٠ شخص ستكون العملية معقدة جدا باستخدام المتغيرات فلابد من استخدام المصفوفات وهي مجموعة من المتغيرات التي تحمل نفس الاسم لكن برقم مختلف.

صيغة حجز المصفوفة:

```
dim array name(maximum number of variables)
```

حجز مصفوفة مكونة من خمس متغيرات تحمل الاسم numbers

```
dim numbers(4)
```

في هذه الحالة تم انشاء ٥ متغيرات في الذاكرة هي:

Numbers(0)
Numbers(1)
Numbers(2)
Numbers(3)
Numbers(4)

كل موقع من هذه الـ ٥ مواقع يمثل حيز يمكن خزن أي عدد فيه و لخزن عدد ما في المتغير الرابع..

```
numbers(3)=6
```

و لخزن اسم 10 اشخاص

```
dim name$(9)
for I=0 to 9
input a$
name$(I)=a$
next I
print "now the names are saved"
input "tell me the number of the person you want to see his name ";n
print name$(n)
```

لاحظ انه يجب ان يكون اسم متغير المصفوفة من النوع المقطعي عندما تكون البيانات المراد خزنها عبارة عن مقاطع نصوص.

٢.المصفوفات ببعدين:

```
dim array name(rows,columns)
```

الـ rows هي الصفوف و الـ columns هي الاعمدة.

Position (0,0)	Position (0,1)	Position (0,2)
Position (1,0)	Position (1,1)	Position (1,2)
Position (2,0)	Position (2,1)	Position (2,2)

مصفوفة 2*2

نستنتج هنا انه يمكن عمل جداول من المصفوفة بالبعدين وكما تلاحظ الجدول اعلاه لاحظ ان الصفوف تزداد نحو الاسفل و الاعمدة تزداد نحو اليمين و يمكنك ايضا ان تعطي قيمة مقطعية او عددية (حسب اسم المصفوفة) ,اما لماذا نستخدم for\next لادخال قيم المصفوفة و ذلك للسرعة فلايعقل ان نعاملها كالمتغيرات العادية وهنا ستكون هناك حلقتين for\next خارجية و داخلية,حيث ستنفذ الحلقة الداخلية حسب عدد مرات الحلقة الخارجية.

في المثال القادم سنقوم بملئ المصفوفة بالرقم 1 وذلك لتوضيح كيفية ملئ مصفوفة ثنائية البعد.

```
Dim a(2,2)
For I=0 to 2
For j=0 to 2
A(I,j)=1
Next j
Next I
```

الحلقة التكرارية الداخلية سوف تتكرر ٣ مرات وبذلك فسوف تمتلئ الصفوف و الاعمدة بالرقم 1.
ربما قد حجزت مصفوفة و تريد بعد ذلك تغيير ابعادها يمكنك ذلك من خلال الامر redim ولكن سوف تحذف كل محتويات المصفوفة.

```
Dim a(10)
For I=1 to 10
A(I)=1
Next I
Redim a(11)
For I=1 to 11
Print a(I)
Next I
Print "program finished"
End
```

هناك دالة مدمجة في هذه اللغة تمكنك من ترتيب محتويات المصفوفة حسب الحروف الابجدية (اذا كانت المحتويات مقاطع نصية) أو تمكنك من ترتيب الاعداد من الاكبر للاصغر أو بالعكس و يتم ذلك عن طريق الدالة sort :

```
sort array(),start,end,column
```

حيث الـarray هو اسم المصفوفة المراد تغيير ترتيبها,و start هو رقم يمثل رقم الصف(row) للعنصر المراد البدء منه للترتيب و end يمثل رقم الصف لاخر عنصر يشمله الترتيب و الاختيار الثالث column يستخدم فقط مع المصفوفات ثنائية البعد ويمثل رقم العمود column المراد ترتيبه.والبرنامج التالي يوضح الفكرة:

```
dim a$(3)
for I=1 to 3
read b$
a$(I)=b$
next I
sort a$(),1,3
print "after sorting"
for I=1 to 3
print a$(I)
next I
data "window","copybook","house"
```

أما لعكس الترتيب فيجب عكس ترتيب الرقمين start و end في صيغة الدالة sort أي يمكن عكس ترتيب الكلمات في البرنامج السابق عن طريق تغيير جملة الترتيب sort الى الجملة التالية:

```
sort a$(),3,1
```

جملة read\data:

read و data لم تختلف عن لغة Basic حيث تحوي الجملة read على متغيرات تستمد القيم العددية او المقطعية من البيانات في جملة data و يجب ان تحاط الثوابت المقطعية بعلامات الاقتباس.

```
Read a$,b
Print “language “;a$
Print “price to buy from internet “;b;” dollars”
Data “liberty basic language”,30
```

ويمكن استخدام الحلقات التكرارية لقراءة عدد معين من البيانات و تستمر الحلقة التكرارية الى عدد مرات قراءة البيانات.

```
For I=1 to 2
Read a$,b
Print “language “;a$
Print “price to buy from internet “;b;” dollars”
Next I
Data “liberty basic language”,30,”visual basic language”,100
```

ويمكن استخدام الجملة restore لاعادة قراءة البيانات من جملة data وبما ان هذا الكتاب يفترض انك قد تعلمت لغة Basic فلن يشرح هذا الامر بصورة مفصلة و لتوضيح ذلك ادرس هذا المثال المأخوذ من ملف الـ help المرفق مع لغة Liberty Basic :

```
'show me my data in all uppercase
   while string$ <> "end"

    read string$
     print upper$(string$)
   wend
   string$ = ""  'clear this for next while/wend loop

   'now reset the data reading to the second part
   restore [partTwo]

   'show me my data in all lowercase
   while string$ <> "end"
     read string$
     print lower$(string$)
   wend

   data "Sally", "Sells", "Sea", "Shells", "By", "The", "Sea",  "Shore"
 [partTwo]
   data "Let's", "Do", "Only", "This", "A", "Second", "Time", "end"

   end
```

الدوال الرياضية:

كثير من هذه الدوال موجودة في لغة basic .
في الامثلة القادمة N يمثل أي عدد.
SQR(N): الجذر التربيعي للعدد.
EXP(N): ايجاد قيمة e^N حيث e تساوي 2.7182818
LOG(N): اللوغاريتم الطبيعي للعدد.

INT(N):ازالة الاعداد بعد الفاصلة العشرية.
MAX(N1,N2) :القيمة العائدة تمثل الرقم الاكبر بين الرقمين.
MIN(N1,N2) :القيمة العائدة تمثل الرقم الاصغر بين الرقمين.
RND(N) :تعيد قيمة عشوائية بين الصفر و الواحد.
برنامج لطباعة ارقام عشوائية من 1-10

```
FOR I=1 TO 10
PRINT INT(RND(1)*10)+1
NEXT I
```

قد تكون هناك مشكلة في توليد نفس الارقام العشوائية في كل مرة لذلك يجب استعمال الامر RANDOMIZE(N) حيث N هنا رقم بين ٠ و ١ و يجب استخدام القيم 0.7 مثلا
البرنامج التالي سيطبع نفس الارقام كل مرة ينفذ فيها.

```
RANDOMIZE 0.5
FOR I=1 TO 10
PRINT INT(RND(1)*100)
NEXT I
```

الدوال المثلثية:

```
SIN(N)
COS(N)
TAN(N)
```

وهناك دوال اخرى تسمى الـ hyperbolic functions هي:

```
ASN(N)
ACS(N)
ATN(N)
```

في الرياضيات ASN تسمى sinh و ACS تسمى cosh و ATN تسمى tanh .

من الممكن ان يكون هناك رقم عبارة عن مقطع من النصوص ولكن لايمكن اجراء العمليات الحسابية عليه و لكن يمكن تحويله الى عدد مرة اخرى عن طريق الدالة VAL ومثال على استعمال VAL:

```
N$="2"
N2=VAL(N$)
PRINT N2+4
```

أي انه تم تحويل مقطع من النصوص الى عدد و لايمكن اجراء هذه العملية الا على النصوص التي اصلها رقم.
وهناك دالة اخرى تعمل بصورة عكسية عن VAL هي الدالة STR$ حيث تتمكن من تحويل الاعداد الى نصوص.

```
W$=STR$(2)
PRINT W$
```

وهناك دالة للتحويل من الـ HEXADECIMAL الى الـ DECIMAL هي
HEXDEC("VALUE") و VALUE هنا مقطع نصوص يمثل رقم بالنظام الستعشري.
اما دالة تحويل الـ DECIMAL الى HEXADECIMAL هي DECHEX$(NUMBER) حيث NUMBER هو العدد المراد تحويله الى النظام الستعشري.

قد تتسائل هل يمكن ادخال معادلة في وقت تنفيذ البرنامج و تعويض عدد فيه ثم حساب الناتج,لقد وفرت لغة Liberty Basic ذلك حيث وفرت دالتين تقومان بنفس العمل و لكن احدهما تكون القيمة العائدة منها هي مقطع نصي و الاخرى القيمة العائدة منها

عبارة عن عدد.الدالة التي تكون قيمتها العائدة هي مقطع نصي هي EVAL$ و الاخرى EVAL .

```
INPUT Y
CODE$="Y+1"
PRINT EVAL(CODE$)
PRINT EVAL$(CODE$)
```

النتيجة نفسها في الحالتين لكن الاولى عبارة عن عدد و الاخرى عبارة عن مقطع نصي.
يستفاد من هذه الدوال في برامج رسم الدوال.

وهناك دالتين لاظهار التاريخ و الوقت هما date$ و time$.
استخدام date$:

```
print date$()  لطباعة التاريخ
print date$("mm/dd/yyyy")لتحديد شكل طباعة التاريخ
print date$("mm/dd/yy")
print date$("yyyy/mm/dd")
print date$("days")  عدد الايام من ١٩٠١\١\١
```

```
print date$("تاريخ")
```

يمثل "تاريخ" في الجملة السابقة أي تاريخ فناتج البرنامج سيكون طباعة عدد الايام من ١٩٠١\١\١ الى التاريخ المعطى في الصيغة.
الناتج هنا هو التاريخ الذي عدد الايام منه الى (عدد الايام من ١٩٠١\١\١)print date$ ١٩٠١\١\١ يساوي عدد الايام في الصيغة اعلاه.

اما time$ فتستعمل هذه الصيغ:

```
print time$()  الناتج هو الوقت الحالي
print time$("seconds")  عدد الثواني من منتصف الليل الى الوقت الحالي
print time$("ms")  عدد الـ(ملي ثانية) من منتصف الليل الى الوقت الحالي
print time$("milliseconds")  نفس الناتج السابق
```

انشاء الدوال بواسطة المبرمج user defined functions:

في لغات البرمجة الحديثة يكون البرنامج عبارة عن اجزاء وقد تحتاج الى حساب بعض بعض التعابير الرياضية لذلك يجب استخدام الدوال في هذه الحالة فيكفي ان تستدعي الدالة و تمرر لها القيم المراد حسابها في الدالة,لذلك فالدوال لها parameters أي متغيرات وتمرر القيم الى الدالة لحساب النتيجة عن طريق اسم الدالة,فالدالة SQR هي دالة موجودة ضمن لغة Liberty Basic وعندما يقوم المبرمج بكتابة :

```
a=sqr(4)
```

ستستدعي هذه اللغة هذه الدالة و تمرر لها القيمة ٤ لتقوم الدالة بحسابها و اعادتها الى المتغير a .
اغلب الاحيان تكتب الدوال في نهاية البرنامج أي اسفل الـ labels ,وطريقة تكوينها تتبع الصيغة التالية:

```
function function name(variables of the function)
function name=mathematical calculations on parameters
end function
```

مثال على الدوال:

```
function squareroot(a)
squareroot=a^0.5
end function
```

اسم الدالة هنا هو squareroot .

في أي مكان من البرنامج الرئيسي يمكن ان تحسب الجذر التربيعي لأي عدد عن طريق كتابة :

```
b=squareroot(4)
print b
```

هنا سيحسب الجذر التربيعي للعدد ٤,وكذلك يمكن ان تحوي الدالة على اكثر من parameter توضع بينها فواصل.

كل متغير يستخدم داخل الدالة يكون مخفيا عن البرنامج الرئيسي و بالعكس,أي لو استخدمت متغيرين لهما نفس الاسم احدهما خارج الدالة و الاخر داخل الدالة سيعتبران متغيرين مختلفين.فمثلا لو كتبت في البرنامج الرئيسي أي خارج الدالة y=4 ثم قمت بكتابة print y داخل الدالة سيكون الناتج لايساوي ٤ أي ان المتغير y داخل الدالة يختلف عن خارج الدالة,ولحل هذه المشكلة هناك نوع من المتغيرات يسمى global أي يمكن ان تستعمل في أي مكان من البرنامج حتى في الدوال المختلفة و يجب تعريف هذه المتغيرات في بداية البرنامج.

```
global y
global numbers
```

المثال السابق جزء من برنامج يستخدم المتغيرات من نوع global ففي هذه الحالة المتغيرين y و numbers يمكن استخدامهما في أي مكان من البرنامج أي ان قيمتهما لن تكون مخفية عن بقية البرنامج.

ان هذه هي الطريقة العادية في تمرير المتغيرات للدوال و هي تسمى passing by value أي ان التغيير الذي سيجري على المتغيرات داخل الدالة لن تتأثر به نفس المتغيرات الموجودة خارج الدالة و لهذا السبب يوجد نوع اخر من تمرير المتغيرات يسمى passing by reference و فيما يأتي مثال على نوعي التمرير:

التمرير passing by value :

```
print "program will calculate the square of any integer.."
input n
print square(n)
print "value of n after the function is ";n
end

function square(a)
a=int(a)
square=a^2
end function
```

في البرنامج اعلاه جرب ادخال عدد غير صحيح مثل 4.6 سيتم الحساب باعتبار 4 هي الرقم المراد حساب مربعه نتيجة استخدام الدالة int في الدالة و هذا النوع من تمرير المتغيرات يسمى passing by value و فرقه عن الـ passing by reference ان المتغير بعد تنفيذ الدالة سيحمل عدد صحيح دائما (في البرنامج السابق) كما في البرنامج التالي:

```
print "program will calculate the square of any integer..":input n
print square(n)
print "value of n after the function is ";n
```

```
end

function square(byref a)
a=int(a)
square=a^2
end function
```

تستطيع تمييز الـ passing by reference بوجود كلمة byref امام بعض أو كل متغيرات الدالة مثلا لو كانت هناك دالة لها متغيرين و نريد استخدام هذا النوع من التمرير سيكون شكل تعريف الدالة:

```
function greaternumber(byref a,byref b)
```

ستلاحظ من البرنامج اعلاه ان القيمة للمتغير n المطبوعة بعد تنفيذ الدالة ستكون قيمة معدلة و تتوقف على العمليات الرياضية التي حصلت داخل الدالة على المتغيرات التي تم تمريرها للدالة فمثلا لن يتذكر البرنامج القيمة التي ادخلت في بداية البرنامج بل سيتذكر القيمة المعدلة من خلال الدالة int في البرنامج اعلاه.

الـ subroutines :

هي عبارة عن مقاطع من البرامج تستعمل عندما نريد تكرار شئ معين مثلا كعرض عبارة معينة على الشاشة مثلا وتتميز بانها يمكن ان تستقبل متغيرات مثل الدوال و تقوم بالتصرف تبعا لذلك.ولكتابة الـ subroutines :

```
sub name of subroutine    parameters(optional)
. . .
. . .
end sub
```

وغالبا ماتوضع الـ subroutines في نهاية البرامج بحيث ان البرنامج لايمر بها و انما يتم استدعاؤها حسب الحاجة وتنطبق عليها كل قواعد الـ global و الـ byref وغيرها من القواعد التي تنطبق على الدوال.

ويتم استدعاء الـ subroutine كالتالي:

```
call subroutine name   passing values
```

و البرنامج التالي يوضح هذا:

```
print "this is a subroutine example"
print "now you will write a word and how many times it would be repeated"
input "what is the word? ";w$
input "how many times to repeat this word? ";n
call typeandrepeat w$,n
print
print "finished typing"
wait

sub typeandrepeat  word$,number
for I=1 to number
print word$
next I
end sub
```

ان عملية تمرير المتغيرات تشبه تمرير المتغيرات في الدوال و التنفيذ في الـ subroutine عندما ينتهي من عملية الطباعة في المثال السابق سيعود الى السطر الذي يلي السطر الذي استدعي منه الـ subroutine.

ولايمكن استخدام الاقواس لوضع متغيرات الـ subroutine فيها.

ويمكن استخدام الـ passing by reference عن طريق وضع byref امام متغيرات الـ subroutine المراد تمريرها بهذه الطريقة.

جملة timer:

قد تحتاج في بعض البرامج و خاصة برامج الالعاب الى تنفيذ بعض المهام كل فترة زمنية معينة لهذا تستخدم الـ timer حيث تحدد من خلاله الـ label المراد تكراره كل فترة زمنية وهذه الفترة الزمنية تكون بوحدة الـ(ملي ثانية milliseconds) حيث الثانية الواحدة تساوي 1000 ملي ثانية.والبرنامج التالي يوضح الـ timer :

```
print "this is a timer program"
input "type a word here to be typed every second? ";w$
timer 1000,[type]
wait

[type]
print w$
wait
```

في هذه الحالة سيتم طباعة الجملة التي اعطيتها للبرنامج كل ثانية لمدة غير محدودة لكن يمكن تحديد المدة أي تعطيل الـ timer بعد هذه المدة ويتم تعطيل الـ timer من خلال جعل الفترة الزمنية تساوي 0 .مع الملاحظة ان لغة Liberty Basic تستطيع استعمال timer واحد فقط.

```
numbers=0
print "this is a timer program"
input "type a word here to be typed every second? ";w$
timer 1000,[type]
wait

[type]
numbers=numbers+1
if numbers>=3 then timer 0
print w$
wait
```

في المثال اعلاه لقد سمحنا للـ timer ان يطبع الجملة ثلاث مرات فقط من خلال متغير قمنا بزيادته كل فترة زمنية أي كل ثانية في البرنامج اعلاه,و قمنا باختبار المتغير فيما اذا كان قد وصل الى القيمة 3 او لم يصل,فاذا وصل الى القيمة 3 او اكبر من هذه القيمة نقوم بتعطيل الـ timer بجعل قيمته تساوي صفر.

التعامل مع النصوص:

التعامل مع النصوص يشمل استخراج جزء من النصوص لاغراض مختلفة فمثلا للبحث عن كلمة في ملف معين يجب معرفة طول الكلمة و حروفها و بذلك يمكن ايجادها.ويشمل ايضا التأكد من وجود حرف او مجموعة حروف في جملة او ملف و غير ذلك مع العلم انه هذه التقنيات مهما كانت بسيطة فهي تعتبر من الامور المهمة في حذف اجزاء من ملفات الحاسبة و اضافة اجزاء عليها كما هو الحال في برنامج Microsoft Word مثلا عند استخدام تقنية البحث عن النصوص و استبدالها داخل ملف.وكما علمنا سابقا ان كل امر من اوامر الحاسبة يمكن ان يرجع قيمة معينة و هنا ستكون القيمة ربما 1 اذا جرت العملية صحيحة و 0 اذا كانت العملية خاطئة او قيمة اخرى تدل على عدد الحروف و غيره.

شفرة ASCII :
كل مفتاح و حرف في الحاسبة له رقم معين يسمى هذا الرقم شفرة ASCII وهي اختصار لـ American Standard Code For Information Interchange وكل حرف له رقم ASCII وتمتد هذه الرموز من ٠-٢٥٥ وتشمل المفاتيح الاخرى مثل enter,space وغيرها.
بعض الاوامر في هذه اللغة تحوي على الرمز ($) وبعضها لا تحتوي و للتفريق بين هاتين المجموعتين ,ان كل امر من الاوامر تكون القيمة العائدة منه عبارة عن مقطع حرفي فهذا الامر يجب ان يحوي على الرمز ($) واذا كانت القيمة العائدة ليست حروف و لكن عبارة عن اعداد فلا يحتوي هذا الامر على الرمز ($).
ويجب قبل البدء بشرح التعامل مع النصوص شرح كيفية خزن النصوص داخل المتغير المقطعي.
مثلا المتغير $word يحوي الجملة hello world فطول هذه الجملة لايحسب بعدد الحروف بل يحسب بعدد المسافات حتى المسافات الفارغة لها سعة تساوي سعة المساحة التي يحتلها حرف فالجملة السابقة سيكون طولها أي بمعنى اخر عدد حروفها يساوي 11 حرف و موقع الحرف w مثلا هو 8 وتنطبق هذه القاعدة على الملفات ايضا فعند كتابة ملف من نوع text في برنامج Microsoft Notepad وحتى اذا كتب فيه حرف واحد مثلا لكن مع وجود مسافات فارغة في البداية سيكون حجم الملف يساوي عدد المسافات المتروكة فارغة الى اخر حرف في الملف و تقاس المسافة الواحدة فارغة كانت ام تحوي على حرف بوحدة الـ Byte حيث سعة الجملة السابقة hello world هي 11 Byte .

الامر chr$:
يقوم بطبع الحرف المقابل للرقم حسب نظام ASCII

```
print chr$(77)
```

الامر asc:
يقوم بطبع الرقم المقابل للحرف حسب نظام ASCII أي عكس الامر السابق:

```
print asc("M")
```

الامر instr:
هذا الامر هو مختصر in string ويختص بالبحث عن مقطع معين او حرف واحد داخل متغير مقطعي او جملة معينة,و القيمة العائدة تكون 0 اذا كان المقطع المراد البحث عنه غير موجود في الجملة,و القيمة العائدة قد تكون أي قيمة شرط ان لاتساوي 0 اذا وجد المقطع المراد البحث عنه في المقطع الاصلي.
وصيغة هذا الامر هي:

```
instr (string1,string2,starting)
```

حيث:
string1 هي الجملة المراد البحث فيها عن المقطع string2 والبحث يبدأ من الموقع المحدد في starting ,و starting هو شئ اختياري أي تستطيع وضعه ضمن الصيغة او حذفه من الصيغة.

```
W$="hello world"
Print instr(W$,"d")
```

وهذا برنامج اخر..

```
print instr("hello world","w",8)
```

انتبه الى ان اسماء المتغيرات هي case-sensitive أي تختلف اذا كانت الحروف المستعملة كبيرة او صغيرة فالمتغير W$ ليس نفس المتغير w$ حيث ان الاول في

حالة الاحرف الكبيرة و الثاني في حالة الاحرف الصغيرة وسيقوم محرر هذه اللغة بتنبيه المبرمج في حالة وجود هذه الحالة عن طريق كتابة similar variables اسفل النافذة.

الامر len:
القيمة العائدة تساوي عدد الحروف التي يتكون منها المقطع الموجود في ضمن هذه الجملة.

```
N=len("hello world")
Print N
```

برنامج اخر..

```
w$="hello world"
print len(w$)
```

الامر $left:
الصيغة

```
left$(string,number)
```

حيث string هي مقطع نصي و number تمثل عدد الحروف المراد استنساخها من يسار الـ string.
مثال:

```
print left$("hello world",5)
```

برنامج اخر..

```
w$="hello world"
n=1
return$=left$(w$,n)
print return$
```

الامر $right:
نفس الامر $left لكن يقوم باستنساخ الحروف من يمين المقطع string .
البرنامج:

```
w$="hello world"
n=1
return$=right$(w$,n)
print return$
```

قد تتسائل الان الايوجد امر لاستنساخ أي قراءة بعض المقاطع من أي مكان نحدده من المقطع النصي,لهذا الغرض يوجد الامر التالي وهو $mid.

الامر $mid:
الصيغة

```
mid$(string,index,number)
```

حيث string هو مقطع نصي و index يحدد مكان بداية القراءة من المقطع و الـ number يمثل عدد الحروف أي المسافات المراد قراءتها من الموقع index.
البرنامج:

```
print mid$("hello world",7,5)
```

في البرنامج السابق تكون القيمة العائدة هي world لأن القراءة بدأت من الموقع رقم 7 من المقطع و استمرت الى 5 حروف بعد الموقع رقم 7 .

الامر $lower:
الصيغة

```
lower$(string)
```

حيث string هو مقطع نصي يحتوي على حروف كبيرة و صغيرة فالقيمة العائدة هي النص string لكن مع تحويل حالة الحروف الكبيرة الى صغيرة مع ابقاء الحروف الصغيرة على حالها.
البرنامج:

```
print lower$("Hello World")
```

الامر upper$:
عكس الامر lower$
البرنامج:

```
w$=upper$("Hello World")
print w$
```

الامر trim$:
الصيغة

```
trim$(string)
```

تكون القيمة العائدة هي النص string لكن مع ازالة المسافات الفارغة على يمين و يسار المقطع string.
البرنامج:

```
print "  Hello World    "
print trim$("  Hello World    ")
```

ملاحظة:المتغير المقطعي لايمكن ان يوضع داخل علامات الاقتباس واذا وضع بين علامتي اقتباس سيعتبر مقطع نصي و ليس متغير مقطعي.

```
W$="    Hello World"
Print W$
Print trim$(W$)
```

الامر space$:
الصيغة

```
space$(number)
```

يقوم بترك مسافة فارغة طولها يساوي العدد number وهذا يشابه ضغط مفتاح المسافة space من لوحة المفاتيح بعدد مرات الـ number .
واذا اردت دمج هذا الامر مع المقاطع النصية يمكنك استخدام الرمز (;) كما في البرنامج التالي:

```
print space$(6);"hello";space$(2);"world"
```

ويمكنك ان لا تستخدم هذا الامر فسيكون البرنامج السابق كالتالي:

```
print "      hello  world"
```

و القيمة العائدة من هذا الامر هي مسافة فارغة بنفس القيمة المحددة في number في الصيغة اعلاه.

يمكن التحكم بالنافذة التي تظهر فيها النتائج و التي تسمى Main Window من خلال الجملة mainwin كما في الصيغة التالية:

```
mainwin columns rows
```

مثال:

```
mainwin 60 20
```

حيث columns هي عدد الاعمدة و الـ rows هي عدد الصفوف.

يمكن ان تحدد من اين تريد الكتابة في النافذة Main Window من خلال الامر locate وكما في المثال التالي:

```
locate 10,11:print "hello world"
```

حيث القيمة 10 تمثل قيمة احداثي الـ x و الـ 11 تمثل الاحداثي y .
وهذه الجملة تحدد موقع مؤشر الكتابة من النافذة.

كما انه يمكن استخدام الامر $space بدلا من ضغط المفتاح space من لوحة المفاتيح هناك امر اخر هو الامر tab حيث يستخدم بدلا من الضغط على مفتاح الـ tab من لوحة المفاتيح كما في البرنامج التالي:

```
print "hello"; tab(8) ;"world"
```

حيث الـ tab يقوم بتحريك مؤشر الكتابة (المؤشر الذي تبدأ عنده الكتابة) من البداية الى الموقع الموجود بين القوسين في صيغة الامر tab .

الامر word$:
الصيغة

```
variable=word$(w$,n,d)
```

هذه الامر يقوم باستخلاص الكلمات من متغير مقطعي او مقطع من النصوص,حيث w$ تمثل النص المراد فصل كلماته عن بعضها و n يمثل رقم الكلمة التي تريد فصلها من الجملة ووضعها في المتغير variable و هذا المتغير يجب ان يكون مقطعي,أما d فيمكن ان تكتبها او لا تكتبها فهي اختيارية حيث تمثل الفاصل بين الكلمة و الاخرى ,اذا لم تكتبها سيعتبر الفاصل بين كل كلمة و اخرى هي المسافة الفارغة " " .

```
print word$("liberty basic",1)
```

ان الناتج هنا سيكون الكلمة liberty أي ان البرنامج طبع الكلمة رقم 1,ويمكن ابدال الرقم 1 بالرقم 2 فيكون الناتج الكلمة basic اما اذا كان هذا العدد اكبر من عدد الكلمات او اصغر من 1 فسيكون الناتج (القيمة العائدة) من هذه الجملة هي "" أي لاشئ.
اما اذا كانت الكلمات مفصولة بفواصل (,) فيمكن فصلها عن بعضها كما يأتي:

```
e$=word$("liberty,basic",2,",")
print e$
```

أي ان المحدد لفصل الكلمات هنا هو الفاصلة.

الامر lprint:
له نفس استعمال print لكنه يقوم بتهيئة طباعة الجملة بين علامتي الاقتباس في صيغته على الطابعة من نوع line printer ولكن ليس مجرد كتابة هذا الامر سيؤدي الى طباعة الجملة بين علامتي الاقتباس لكن يجب ارسال المعلومات الى الطابعة و ذلك عن طريق الامر dump .و البرنامج التالي يوضح ذلك:

```
lprint "hello world"
dump
```

حيث يجب كتابة lprint لكل سطر ثم ارسال معلومات السطر الى الطابعة عن طريق الامر dump.
مثلا ملف يتكون من سطرين ,اولهما يحوي على "hello world" و السطر الثاني يحوي على "liberty basic" ,ستكون عملية طباعتهما كالتالي:

```
lprint "hello world"
dump
lprint "liberty basic"
dump
```

أي ان عمل dump هو عملية ارسال كل عبارة قمت بكتابتها ضمن جملة lprint الى الطابعة,يمكن طباعة الجمل حتى بدون استخدام lprint لكن الطباعة قد تكون غير صحيحة.

نظام الملفات:

ماهي الملفات؟
مواقع في الذاكرة يمكن الوصول اليها بسرعة ويمكن كتابة ملاحظات او معلومات فيها أو يمكن ان يكون نوع الملف ملف برنامج أي ليس من نوع كتابة الملاحظات كالذي تكتبه في برنامج Microsoft Word مثل ملفات الـ EXE وهي البرامج التنفيذية التي قد تكون العاب كمبيوتر او قواعد بيانات و غيرها..

كيف نحدد حجم الملف؟
يمكن تحديد حجم الملف عن طريق عدد الـBytes فيه.

ماهي الـBytes؟
كل حرف يكتب في الملف يسمى Byte.

لماذا يسمى Byte؟
لأن كل حرف يتألف من ثمانية bits والـ bit هو رقم اما صفر أو واحد,حيث كل حرف يمثل بالطريقة الثنائية binary بثمانية ارقام صفر و واحد.
المسافة الفارغة أيضا تمثل byte .

لتوضيح الفكرة افرض ان هناك ملف نوع text يحوي المحتويات التالية:

hello world

كم سيكون حجم الملف؟
سيكون ١١ byte.و الـbyte =٨ bit ,حجم الملف بالـ bits هو ٨٨ bit.

افرض ان كل السطر الواحد من برنامج notepad يستوعب ٧٠ byte فكم سيكون حجم الملف التالي؟

hello world
test program

بمجرد وجود سطر ثاني أي ان نهاية الملف لم تكن عند hello world لأن حجم الملف يقاس حتى اخر بايت مكتوب في الملف حتى لو تركت عشرات الاسطر فارغة.حجم الملف هو ٨٢ byte.
يجب فهم موضوع الملفات و خاصة الكتابة في ملف و القراءة منه جيدا لأن لغة Liberty Basic تعتبر ان كل نافذة عبارة عن ملف وستتوضح هذه الفكرة اثناء شرح كيفية تكوين النوافذ.
لكل ملف شئ يسمى رقم الملف او handle ,ويتم التعامل مع الملف من خلال رقمه.

وهناك ثلاثة انواع من الملفات:

١.الملفات التسلسلية.
٢.الملفات العشوائية.
٣.الملفات الثنائية.

ان التعامل مع كل نوع من الملفات متشابه,فقبل كل شئ يجب تكوين الملف ان لم يكن موجودا مسبقا ثم فتح هذا الملف ثم قراءة المعلومات او البيانات من الملف او كتابة معلومات او بيانات في الملف ثم غلق الملف لحفظ التغييرات التي حدثت في الملف.
لايمكن ان يكون هناك ملفين لهما نفس الرقم او الـ handle .

لماذا قد تحتاج للملفات؟
قد تحتاج الى الملفات عند انشاء برامج قواعد البيانات أي انشاء برنامج يخزن معلومات معينة و قد تحتاج البحث عن هذه المعلومات فستستخدم اوامر التعامل مع النصوص التي شرحناها سابقا.او ربما قد تقوم ببرمجة احدى العاب الحاسوب وتريد ان تحفظ مكان وجود الشخصية الرئيسية للعبة و مكان الخصوم في اللعبة و غير ذلك من الاستخدامات المفيدة للملفات.

لماذا هناك ثلاثة انواع من الملفات؟
الملفات التسلسلية تستخدم لحفظ المعلومات وحتى ان كانت قاعدة بيانات تحوي معلومات كثيرة و الملفات العشوائية تستخدم نظام السجلات حيث تستطيع الوصول الى معلومات معينة عن طريق معرفة رقم السجل الذي توجد فيه تلك المعلومات أما الملفات الثنائية فهي من اهم انواع الملفات حيث ان كل الملفات التنفيذية بالامتداد EXE. هي ملفات ثنائية و كل ملفات الصوت WAV. و غيرها من الملفات.

الملفات التسلسلية:
عملية تكوين ملف غير موجود يتم عن طريق جملة open

```
open file name for output as #number
```

حيث file name عبارة عن المسار الكامل للملف مع الامتداد,و number هو رقم الملف أو مايسمى handle .كما في البرنامج التالي الذي يقوم بتكوين الملف test.txt في المسار d:\ .

```
open "d:\test.txt" for output as #1
```

لتكوين الملف يجب غلق هذا الملف لحفظ التغييرات وذلك يتم عن طريق close يليها رقم الملف:

```
close #1
```

وبين هاتين الجملتين ستقوم بكتابة اوامر الكتابة في الملف,ان الكتابة في الملف تشبه عملية فتح برنامج Microsoft Word و فتح ملف من النوع txt. من خلاله و الكتابة داخل هذا الملف ثم حفظ الملف.
سنقوم بكتابة بعض الجمل في هذا الملف ,الجمل التالية يجب ان تكون بين الجملتين السابقتين:

```
print #1,"liberty basic"
print #1,"hello";
print #1,"world"
```

الكتابة في الملف تتم عن طريق الجملة print لكن يجب كتابة رقم الملف في البداية ثم كتابة فاصلة ثم كتابة ماتريد ان تكتبه في الملف مهما كان متغيرا عدديا او مقطعيا او ثابت مقطعي كما في المثال اعلاه.
سيبدو البرنامج السابق كالتالي:

```
open "d:\test.txt" for output as #1
print #1,"liberty basic"
print #1,"hello";
print #1," world"
close #1
```

اذا قمت بفتح الملف و نسيت ان تغلقه و حاولت ان تفتح الملف مرة اخرى فستحصل على خطأ من لغة Liberty Basic لأن كل ملف يجب ان تغلقه باستخدام close .
لو قمت بالانتقال الى الـ Drive D: فستلاحظ وجود الملف test.txt يمكنك ان تشاهد محتواه بالنقر المزدوج بالزر الايسر للفأرة على الملف.
لو قمت مرة اخرى بفتح الملف بنفس الطريقة و غيرت المحتويات ,ستحذف المعلومات القديمة كلها و ستأخذ المحتويات الجديدة مكانها كما في البرنامج التالي الذي سيقوم بالكتابة في الملف السابق مع حذف محتوياته:

```
open "d:\test.txt" for output as #2
print #2,"test number 2"
print #2,"writing new data in file"
close #2
```

انتقل الى الـ Drive D: ثم شغل الملف مرة اخرى ستلاحظ ان محتوياته تغيرت.أي انه عندما تكون صيغة امر open تحوي على العبارة output فهذا يعني ان كل معلومات جديدة تكتب في الملف بعد غلقه تحل محل المعلومات القديمة.
أما اذا اردت ان تضاف المعلومات الحالية الى المعلومات القديمة فيجب ان تبدل العبارة output بالعبارة append فتصبح صيغة الملف كالتالي:

```
open "d:\test.txt" for append as #2
print #2,"test number 2"
print #2,"writing new data in file"
close #2
```

ان append تقوم بتكوين الملف ان لم يكن موجودا,وتقوم باضافة معلومات الى الملف عن طريق print اذا كان موجودا أو ليس موجودا.

حتى الان شرحنا عملية الكتابة الى ملف بنوعيها الـ output و الـ append ,الان سنشرح عملية قراءة معلومات من الملف.
اول خطوة في عملية القراءة من الملف هي فتح الملف ولكن بابدال append أو output بالكلمة input كالتالي:

```
open "test.txt" for input as #1
```

عملية القراءة من الملف هي وضع محتويات الملف في متغير مقطعي لنقوم بقراءته واجراء التعديلات عليه مثل تصحيح الاخطاء فيه او كتابته على الشاشة.وطبعا بعد فتح الملف بهذه الطريقة ستكتب جملة القراءة من الملف للاطلاع على البيانات المخزونة في الملف,وهذه الجمل تتنوع ولكل منها استعمال ما:

جملة input:
الصيغة

```
input #number,variable
```

حيث number هو رقم الملف المراد القراءة منه و الـ variable هو متغير مقطعي يقوم بخزن النص المأخوذ من الملف الذي رقمه number ,والنص المأخوذ هنا هو ليس كل محتويات الملف و لكنه جزء من النص يمتد حتى فاصلة (,) موجودة في الملف.
مثل:

```
input #1,w$
```

في هذه الجملة سيحمل المتغير w$ النص الموجود في الملف الى اول فاصلة (,) في الملف.

جملة line input:
الصيغة

```
line input #number,variable
```

يقوم بقراءة سطر واحد من الملف و اذا كررته مرة اخرى سيقرأ السطر الثاني و يخزنه في المتغير variable و هكذا كلما تكرر سيقرأ السطر التالي و يخزنه في المتغير الموجود في هذه الجملة.و بالطبع ان number هو رقم الملف المراد القراءة منه.
نوع المتغيرات المستعملة هنا هي المتغيرات المقطعية.

جملة input$:
الصيغة

```
variable=input$(#number,n)
```

حيث variable هو المتغير المقطعي الذي سيخزن فيه الجزء من النص الذي ستتم قراءته من الملف,و number هو رقم الملف و n هو عدد الحروف أو الـ Bytes التي ستتم قراءتها من الملف.

جملة inputto$:
الصيغة

```
variable=inputto$(#number,d)
```

حيث variable هو المتغير المقطعي الذي سيتم خزن النص المأخوذ من الملف فيه,و number هو رقم الملف و d هو رمز بين علامتي اقتباس قد يكون مسافة فارغة و قد يكون فاصلة او غيره و هو يمثل الرمز الذي ستتوقف عنده قراءة الملف,أي مثلا اذا وضعت فاصلة بين علامتي اقتباس فقراءة النص من الملف ستتوقف عندما يجد البرنامج فاصلة في الملف.

```
W$=inputto$(#1,",")
```

لنفرض ان محتويات الملف هي liberty basic,is a good programming language فالمتغير W$ و الذي سيحمل النص المأخوذ من الملف سيكون محتواه الجملة liberty basic فقط لأنه صادف وجود فاصلة في محتويات الملف فيتوقف و لايكمل القراءة من الملف.
و البرنامج التالي يوضح استعمال هذه الجمل:

```
'create a sample file

 open "d:\test.txt" for output as #1

 print #1, "liberty basic, is a good programming language"

 close #1



 'INPUT
```

```
open "d:\test.txt" for input as #1

INPUT #1, txt$

print "INPUT item is: ";txt$

close #1


'LINE INPUT

open "d:\test.txt" for input as #1

LINE INPUT #1, txt$

print "LINE INPUT item is: ";txt$

close #1


'INPUT$

open "d:\test.txt" for input as #1

txt$ = INPUT$(#1, 10)   'read 10 characters

print "INPUT$ item is: ";txt$

close #1


'INPUTTO$

open "d:\test.txt" for input as #1

txt$ = INPUTTO$(#1, " ") 'use a blank space as delimiter

print "INPUTTO$ item is: ";txt$

close #1
```

واضافة الى هذا اذا اردت ان تعرض كل محتويات الملف باستخدام الجملة input$
فيجب ان تعرف عدد حروف أي Bytes للملف و لكي تعرف ذلك استخدم الجملة lof و
هي مختصر لـ length of file و الصيغة هي:

```
lof(#number)
```

حيث يمكن استخدامها بدلا من كتابة عدد الحروف المراد قراءتها من الملف لأن القيمة العائدة من هذه الجملة هي عدد الحروف في الملف الذي رقمه number :

```
w$=input$(#1,lof(#1))
```

اضافة الى هذا يوجد هناك جملة تستعمل للتأكد من ان عملية قراءة الملف قد وصلت الى نهاية الملف ام لم تصل و هذه الجملة تكون قيمتها العائدة لاتساوي صفر اذا وصلت قراءة الملف الى نهايته و تكون القيمة صفر اذا لم تصل عملية قراءة الملف الى نهاية الملف.هذه الجملة هي eof و هي مختصر لـ end of file و الصيغة هي:

```
eof(#number)
```

حيث number هو رقم الملف.

والبرنامج التالي سيوضح كيفية استخدام هذا الامر:

```
open "d:\test.txt" for input as #1
while eof(#1)=0
line input #1,t$
print t$
wend
close #1
```

سيستمر تنفيذ الجمل التي بين while و wend الى ان تصبح قيمة eof(#1) لاتساوي صفر أي سيستمر بقراءة سطور الملف سطرا بعد اخر و طبع كل سطر بعد قراءته من الملف الى ان ينتهي من اخر سطر من الملف و في تلك الحالة ستصبح قيمة eof(#1) لاتساوي صفر فتنتهي الحلقة التكرارية ويقوم البرنامج بغلق الملف close #1 .

الملفات الثنائية:

هي الملفات التي يتم التعامل معها على شكل حروف Bytes أي تقوم بكتابة حرف في مكان ما من الملف أو تأخذ حرف ما من مكان ما و هذا المكان في الملف الثنائي Binary File هو مايسمى بمؤشر الملف File pointer وهو المؤشر الذي يدل على الموقع الحالي أي في أي Byte من الملف يتواجد المؤشر(مؤشر الكتابة أو القراءة) أي ان كل عملية كتابة في الملف أو قراءة من الملف ستتم في ذلك المكان الذي يتواجد فيه مؤشر الملف File Pointer .

يتم تكوين ملف ثنائي للقراءة أو الكتابة من خلال الصيغة التالية:

```
open file name for binary as #number
```

حيث file name هو اسم الملف ويمكنك ان تضع له أي امتداد حتى لو لم يكن txt. و الـ number هو رقم الملف.

و مؤشر الملف قبل طباعة أي شئ داخل الملف يكون في الـ Byte رقم صفر و هو أول Byte في الملف الثنائي.

وطباعة البيانات داخل الملف الثنائي تتم عن طريق الجملة print و تستعمل كما في الطريقة السابقة وهنا سيتحرك مؤشر الملف الى مابعد اخر حرف في الجملة المطبوعة في الملف.

Print #1,”basic”

يمكن معرفة موقع مؤشر الملف File Pointer من خلال الدالة loc و هي اختصار لـ location و الصيغة هي :

loc(#number)

حيث number هو رقم الملف,و القيمة العائدة من هذه الجملة هي رقم الـ Byte الذي يتواجد عنده مؤشر الملف File Pointer :

print loc(#1)

ويمكن تحريك مؤشر الملف الى أي مكان ضمن الملف من خلال seek و الصيغة هي:

seek #number,Byte number

حيث number هو رقم الملف و الـ Byte number هو رقم الـ Byte الذي نريد لمؤشر الملف File Pointer أن يتوجه اليه.

اما قراءة المعلومات من الملف الثنائي فتتم من خلال جملة input والصيغة هي:

input #number,variable

حيث number هو رقم الملف و الـ variable هو متغير مقطعي ستخزن فيه القيمة المأخوذة من الملف,أما القيمة المأخوذة من الملف فهي تمتد من موقع مؤشر الملف الحالي الى نهاية الملف,فاذا كان مؤشر الملف في الموقع 4 فستكون القراءة من جميع الحروف (الـ Bytes)من الـ Byte رقم 4 الى الـ Byte الاخير من الملف.

و البرنامج التالي يوضح هذه الجمل:

```
'binary file example
open "d:\myfile.bin" for binary as #myfile

txt$ = "I like programming with Liberty BASIC."
print "Original data in file is:  ";txt$

'write some data to the file

print #myfile, txt$

'retrieve the position of the file pointer
nowPos = LOC(#myfile)

'move the filepointer
nowPos = nowPos - 14
seek #myfile, nowPos

'read the data at the current location
input #myfile, txt$

'print txt$ in mainwin
print "Data at ";nowPos;" is: ";txt$

'move the filepointer
seek #myfile, 2
```

```
'write some data to the middle of the file
print #myfile, "want"
print str$(loc(#myfile) - 2); " bytes written"

'move the file pointer to the beginning

seek #myfile, 0

'read the data
input #myfile, txt$

'print data in mainwin
print "New Data is: ";txt$

'close the file
close #myfile
end
```

مع الملاحظة انه اذا كتبت بعض الحروف في الملف في موقع ما من الملف ثم كتبت حروف اخرى في الملف و في نفس الموقع ايضا فان الحروف الجديدة ستحل محل الحروف القديمة في ذلك الموقع من الملف.

الملفات العشوائية:

الملفات العشوائية هي عبارة عن سجلات تحتوي جدول معين و تم تسميتها بالعشوائية لأن المبرمج أو المستخدم يستطيع الوصول الى المعلومات الموجودة في سجل ما بدون أن يمر على السجلات الاخرى فيكفي وجود رقم السجل,ولكل سجل طول احرف معين أي انه له القابلية على خزن عدد معين من الاحرف (الـ Bytes) و كل حقل من الجدول له القابلية على خزن عدد معين من الاحرف.ويمكن تشبيه الملف العشوائي بالجدول التالي:

Record number	Programming language	price
1	Liberty Basic	30
2	Visual Basic	100
3	Visual C++	100

في الجدول السابق واجهة ملف عشوائي,ان الجدول مقسم الى حقول,وهناك سجلات تحمل معلومات عن منتج ما ,ففي المثال السابق مقارنة بين اسعار لغات البرمجة حيث السجل 1 يحوي معلومات عن لغة Liberty Basic و السجل رقم 2 يحتوي

معلومات عن لغة Visual Basic و السجل رقم 3 يحتوي معلومات عن لغة Visual C++ , يجب تكوين واجهة الملف العشوائي و هي الجدول و يكون ذلك عن طريق الجملة field و الصيغة هي:

FIELD #n,n1 AS var1,n2 AS var2,n3 AS var3

ويمكن ان تستمر الى عدد متغيرات اكثر,حيث n هو رقم الملف المراد فتحه لخزن المعلومات فيه و n1 هو اكثر عدد من الحروف(Bytes) يمكن ان تستعملها للحقل var1 ,و var1 هو متغير قد يكون مقطعي أو عددي و هو يمثل هنا الحقل مثل price في الجدول السابق,وكذلك n2 و n3 هي اعداد الحروف و var2 و var3 هي متغيرات.

اما فتح الملف لكتابة هذه المعلومات فيه فهو كالتالي:

open file name for random as #n len=n1+n2+n3

الصيغة واضحة جدا بعد الاطلاع على انواع الملفات الاخرى ماعدا انه هنا يوجد جملة len=length of file حيث n1+n2+n3 يمثل عدد الحروف في كل الملف أي ان كل سجل من الملف له القابلية على خزن n1+n2+n3 حرف.يجب كتابة جملة open اولا ثم كتابة جملة field .وللتوضيح:

```
open "d:\test.dat" for random as #1 len=22
field #1,20 as name$,2 as price
```

ولادخال المعلومات الى الملف العشوائي أي انشاء سجل(record)جديد يجب ان نمرر القيمة المراد خزنها الى المتغيرات الموجودة في صيغة field ثم اختيار السجل المراد وضعها فيه و لنفرض اننا نريد وضع هذه المعلومات في السجل رقم 1 فتكون تكملة البرنامج السابق هي:

```
name$="Liberty Basic"
price=30
```

في الخطوتين اعلاه قمنا باعطاء قيم للمتغيرات في جملة field لكن المعلومات لم تخزن ضمن السجل و لخزن المعلومات ضمن السجل رقم 1 مثلا يجب كتابة:

```
put #1,1
```

ثم يجب اغلاق الملف close #1

اما للقراءة من الملف فيمكن كتابة البرنامج التالي لقراءة المعلومات من السجل رقم 1 من الملف d:\test.dat

```
open "d:\test.dat" for random as #1 len=22
field #1,20 as name$,2 as price
get #1,1
print "Name is ";name$
print "Price is ";price
close #1
```

حيث get يقوم باخراج المعلومات من السجل ويقوم باسناد متغيرات الـfield بالقيم المخزونة داخل هذا السجل,ففي المثال السابق بمجرد كتابة get #1,1 سيقوم البرنامج باخراج القيم المخزونة في هذا السجل و وضع القيم في المتغيرين name$ و price فيمكن طباعة محتوياتها بعد ذلك.

الامر gettrim :

له نفس استعمال get لكنه يقوم بازالة المسافات الفارغة من يمين و يسار البيانات المأخوذة من الملف العشوائي أي عمل trim لها.

وله نفس صيغة get.

التحقق من وجود الملفات:

يمكن التحقق من وجود الملفات باستخدام الجملة files حيث من الممكن اعطاء مسار ما في هذه الجملة وخزن معلومات هذا المسار في مصفوفة من النوع المقطعي و

ليس العددي و تتضمن المعلومات المخزونة,عدد الملفات في المسار و حجم هذه الملفات ثم معلومات عن كل ملف و مجلد (folder).

```
Dim info$(10,10)
Files "c:\",info$()
```

ستخزن المعلومات في المصفوفة info$,وستخزن المعلومات كالتالي:
المكان 0,0 يحوي عدد الملفات في المسار و المكان 0,1 عدد المجلدات الفرعية و المكان 0,2 اسم الـ Drive و المكان 0,3 هو مسار المجلد الحالي.
وتمتد اسماء الملفات من الموقع 1,0 الى موقع يساوي عدد الملفات أي اذا وجد 29 ملفا فستمتد اسماء الملفات الى الموقع 29,0 أي ان كل موقع من هذه المواقع سيحمل اسم الملف اما الموقع 1,1 هو حجم الملف و 1,2 هو تاريخ انشاء الملف ووقت انشاءه ولكل موقع ملف هذه الصفة أي حجم الملف الذي في الصف row رقم 28 موجود في الموقع 28,1 .

```
Dim info$(100,100)
Files "c:\",info$(
print info$(0,0)
print info$(0,1)
print info$(0,2)
print info$(0,3)
for i=1 to val(info$(0,0))
print info$(i,0),info$(i,1),info$(i,2)
next i
print
print "Sub-Folders"
print
for j=val(info$(0,0))+1 to val(info$(0,0))+1+val(info$(0,1))-1
print info$(j,0)
next j
end
```

ولا يمكن الدخول الى المجلدات الفرعية من خلال هذا للمسار الذي اعطيناه للامر files فالمسار هنا هو c:\ فسيكون ترتيب المعلومات في المصفوفة الناتجة كالتالي:

By:gameprogrammer

مسار المجلد الحالي	المسار المعطى	عدد المجلدات	عدد الملفات 0,0
	الوقت و التاريخ	الحجم	الملف الاول
	=	=	الملف الثاني
	=	=	الملف الثالث
	=	=	وهكذا الى نهاية الملفات في المسار
			وبعد انتهاء الملفات تبدأ اسماء المجلدات
لايوجد معلومات	لايوجد معلومات	لايوجد معلومات	المجلد الاول

=	=	=	المجلد الثاني
=	=	=	وهكذا

يعني اذا فرضنا ان المسار المعطى في جملة files هو c:\ و لنفرض ان عدد الملفات في هذا المسار هو 29 و عدد المجلدات هو 12 فستكون المصفوفة الناتجة كما يأتي:
الموقع 0,0 يحوي ٢٩ ,الموقع 0,1 يحوي ١٢ ,الموقع 0,2 يحوي c:\ ,الموقع 0,3 لايحوي شئ.
الموقع 1,0 يحوي اسم الملف الاول,الموقع 1,1 حجم الملف الاول,الموقع 1,2 تاريخ ووقت انشاء الملف الاول.
الموقع 2,0 يحوي اسم الملف الثاني,الموقع 2,1 حجم الملف الثاني,الموقع 2,2 تاريخ ووقت انشاء الملف الثاني.
وهكذا الى الموقع 29 حيث تنتهي اسماء الملفات و معلوماتها.فتبدأ المواقع التالية:
الموقع 30,0 اسم المجلد الاول
الموقع 31,0 اسم المجلد الثاني
وهكذا الى الموقع 41,0 حيث يحوي اسم المجلد الاخير.وبقية المصفوفة تبقى فارغة.
وكذلك يمكن البحث عن ملف ما..

```
files "c:\","autoexec.bat",info$(
```

فهنا سيقوم البرنامج بالبحث عن الملف autoexec.bat في المسار c:\ فاذا وجده سيضع المعلومات في المصفوفة info$() وبالترتيب الذي شرحناه سابقا,حيث يمكن معرفة اذا كان الملف موجود او لا وذلك عن طريق الموقع 0,0 من المصفوفة فاذا كان يساوي 1 أي ان عدد الملفات التي وجدها و التي تحمل هذا الاسم هي واحد فالملف موجود و اذا كان يساوي 0 فالملف غير موجود.و تجد اسم الملف في الموقع 1,0 و معلومات عنه في الموقعين 1,1 و 1,2 .

اذا كنت تصمم برنامج بلغة Liberty Basic وكنت قد حفظت البرنامج في المسار D:\ يمكنك من معرفة المسار الحالي للبرنامج الذي تكتبه من خلال الجملة DefaultDir$ وهذا المتغير يجب مراعاة الحروف الكبيرة و الصغيرة فيه فلا يجب ان تكتبه هكذا defaultdir$ لأنه متغير من النوع global و هو مدمج مع لغة Liberty Basic وهو يحمل المسار الحالي للملف أو البرنامج الذي تقوم بكتابته في محرر نصوص لغة Liberty Basic .

لتشغيل الملفات باستخدام البرامج مثل تشغيل الملف test.txt باستخدام برنامج notepad.exe يمكن استخدام الامر run و الصيغة هي:

```
run string expression,mode
```

حيث string expression قد يكون مسار البرنامج المراد تشغيله,أو مسار البرنامج المراد تشغيله مع الملف المراد عرضه من خلال هذا البرنامج أما الـ mode فهو يمثل كيفية عرض نافذة البرنامج المراد تشغيله وهو اختياري أي يمكن ان تكتبه او لا.

```
Run "qbasic.exe"
Run "notepad test.txt",MINIMIZE
```

ويمكن كتابة أي من الجمل التالية في مكان الـ mode

```
HIDE
SHOWNORMAL  (this is the default)
SHOWMINIMIZED
SHOWMAXIMIZED
SHOWNOACTIVE
SHOW
MINIMIZE
```

SHOWMINNOACTIVE
SHOWNA
RESTORE

تغيير اسم ملف باستخدام الامر name:
الصيغة

name old name as new name

ومثال على ذلك:

name "d:\test.txt" as "d:\test2.txt"

حيث تم تغيير اسم الملف من test.txt الى test2.txt.

انشاء مجلد جديد باستخدام جملة mkdir :
ومثال ذلك انشاء مجلد اسمه test2 في المسار d:\ في البرنامج التالي:

result=mkdir("d:\test2")
print result

القيمة العائدة تكون 0 اذا تكون المجلد بصورة صحيحة.وهذا المجلد يكون فارغا.

حذف المجلد باستخدام rmdir :

r=rmdir("d:\test2")

اذا تم حذف المجلد بصورة صحيحة ستكون القيمة العائدة تساوي 0.

حذف الملفات باستخدام الامر kill:
الصيغة

kill file path and file name with extension

مثال على ذلك:

kill "d:\test.txt"

حذف الملف test.txt من المسار d:\ .
اما اذا كان الملف المراد حذفه موجود مع ملف البرنامج الحالي وفي نفس المسار فيمكن كتابة اسم الملف مع الامتداد (extention) بدون المسار.

Kill "test.txt"

او مثلا ان الملف موجود في مجلد اسمه test وهذا المجلد موجود في نفس مسار البرنامج الحالي فيمكن حذف هذا الملف كالتالي:

kill "test\test.txt"

لمعرفة اسماء سواقات الاقراص الموجودة في الحاسبة هناك متغير من نوع global مدمج مع لغة Liberty Basic يحمل اسماء هذه السواقات,وهذا المتغير يحمل الاسم Drives$.

print Drives$

الناتج من هذه الجملة سيكون طبع سواقات الاقراص على النافذة Main Window .

في الدروس السابقة اقتصر العمل في هذه اللغة على النافذة Main Window وهذه النافذة تمثل المكان الذي ستطبع فيه الناتج,اذا كان استخدامك لهذه اللغة استخداما عاديا أي ليس احترافيا فيمكن ان تكتفي بهذا القدر من المعلومات عن هذه اللغة دون تعلم انشاء النوافذ او الرسم او استدعاء مكتبات الربط الديناميكية DLL للاستفادة منها في انجاز وظائف غير متوفرة في هذه اللغة,حتى الان لقد تعلمت كيفية كتابة برنامج لغة Basic مع بعض الاضافات في بيئة نظام Windows و ليس في بيئة نظام DOS كما في أغلب لغات الـ Basic السابقة,اما القوة الحقيقية في البرمجة بهذه اللغة فتبدأ بعد

هذا الدرس و ذلك بالتعرف على انواع النوافذ في نظام Windows و الرسم و برمجة العاب الحاسوب الثنائية البعد و التعامل مع الاصوات وغيرها من قوة البرمجة في بيئة نظام الـ Windows .

انواع النوافذ في لغة Liberty Basic:
بالاضافة الى نافذة Main Window توجد نوافذ اخرى.

نافذة الـ text :تمكنك هذه النافذة من كتابة بعض النصوص و التعامل معها كما يمكن التعامل مع برامج التعامل مع النصوص مثل Notepad المرفق مع نظام Windows ,و التعامل مع هذه النوافذ يشمل عرض المعلومات من ملف مثلا في هذه النافذة و استنساخ اجزاء من النصوص و غيرها.ويتم فتح نافذة الـ text كالتالي:

```
open window title for text as #handle
```

حيث window title هو عنوان النافذة المراد فتحها و عرضها على الشاشة,و الـ handle تمثل رقم أو مايسمى مقبض النافذة (window handle) وقد يكون رقم او اسم.
ويتم التعامل مع النوافذ بنفس طريقة التعامل مع الملفات تقريبا.

وفيما يأتي الاوامر المختلفة التي قد توجهها الى النافذة من نوع text:

```
print #handle,"!cls"
```

لمسح نافذة الـ text من كل محتوياتها.

```
Print #handle,"!contents varname$"
Print #handle,"!contents #handle"
```

العبارة الاولى تقوم بملئ نافذة النصوص text بمحتويات المتغير المقطعي varname$ و العبارة الثانية تقوم بملئ نافذة النصوص بمحتويات الملف الذي قيمة الـ handle له هي قيمة الـ handle في هذه الصيغة.مثلا:

```
open "textwindow" for text as #1
open "d:\test.txt" for input as #2
print #1,"!contents #2"
wait
```

المثال السابق مع انه فيه خطأ ما عند اغلاقه لكنه يأخذ محتويات الملف test.txt و يعرضها في نافذة النصوص و هذا اسهل بكثير من عرض محتويات الملف بالطريقة التي شرحناها سابقا أي اخذ المعلومات من الملف سطرا بعد اخر.بعد عدة اوامر سنأخذ كيفية اصلاح الخطأ في البرنامج السابق.

```
Print #handle,"!contents? string$"
```

هذا الامر سيستنسخ النص الموجود في نافذة النصوص و يضعه في المتغير المقطعي الموجود في صيغة هذا الامر.

```
Print #w,"!contents? t$"
```

```
Print #handle,"!copy"
```

استنساخ النص المختار من نافذة النصوص و ارساله الى Windows في مكان يسمى الـ Clipboard وهو يستخدم لخزن النصوص و الملفات المستنسخة.

```
Print #handle,"!cut"
```

اقتطاع النص المختار من نافذة النصوص و ارساله الى الـ Clipboard.

Print #handle,"!font fontname pointsize"
اختيار نوع الخط وحجم الخط للكتابة في هذه النافذة (نافذة النصوص).
Print #w,"!font Times_New_Roman 10"

Print #handle,"!insert varname$"
اضافة محتويات المتغير varname$ الى مكان وجود مؤشر الكتابة في نافذة النصوص مع ابقاء هذا النص المضاف مضللا.

Print #handle,"!line n string$"
يقوم باستنساخ السطر الذي رقمه محدد بالرقم n ووضع هذا النص في المتغير string$. مثلا قراءة السطر السادس من نافذة النصوص تم كما يأتي:
print #w,"!line 6 w$"
حيث w$ ستحتوي على محتويات السطر السادس من نافذة النصوص.

Print #handle,"!lines countvar"
سيقوم هذا الامر بحساب عدد السطور و يضع هذا العدد في المتغير العددي countvar

Print #handle,"!modified answer$"
عند كتابة بعض الكلمات في نافذة النصوص قد تريد ان تنبه لمستخدم البرنامج ان يحفظ هذه المعلومات في ملف ما و لذلك وللتأكد من ان تغييرا قد حصل على محتويات نافذة النصوص يمكن استخدام هذا الامر و ستكون قيمة المتغير answer$ اما true أي قد حصل تغيير في النص او false أي لم يحصل تغيير في النص.وهذا ما تلاحظه في اغلب برامج الطباعة حيث بمجرد فتح ملف ما و كتابة كلمات اخرى فيه و محاولة اغلاق البرنامج فالبرنامج قبل اغلاقه سيسأل اذا ماكنت تريد حفظ التغييرات التي اجريتها على الملف و هنا و من خلال هذا الامر يمكن ان تستفاد من هذه الصفة.

Print #handle,"!paste"
كل نص عملت له copy او cut فهو يتقبل عملية الـ paste فمن خلال هذا الامر تستطيع لصق النص المستنسخ او المقتطع من نص اخر الى مكان مؤشر الكتابة الحالي في نافذة النصوص.

Print #handle,"!select column row"
من خلال هذا الامر تستطيع اختيار أي نص تريد بكتابة الـ column أي العمود و الـ row أي الصف الذي تريد اختياره واذا اردت استعمال متغيرات فلا توضع داخل الاقواس بل تكون بالصيغة التالية:
print #w,"!select ";column;" ";row

Print #handle,"!selectall"
هذا الامر يؤدي الى اختيار كل النص الموجود في نافذة النصوص.

Print #handle,"!selection? selected$"

هذا الامر يؤدي الى استنساخ النص المختار حاليا و وضعه في المتغير المقطعي $selected فبذلك يمكن ان نعرف ماهو النص الذي اختاره مستخدم البرنامج حاليا.

Print #handle,"!setfocus"

نقل التركيز الى نافذة النصوص,في نظام Windows تكون النافذة الحالية التي تستعملها هي النافذة التي تكون فعالة Active أي ان لها focus أي تركيز و عندما تريد العمل على نافذة اخرى فيكفي النقر على النافذة الاخرى و بذلك ينتقل التركيز (focus) الى النافذة الاخرى و تصبح هي النافذة الفعالة Active.

Print #handle,"!trapclose [branchlabel]"

قبل عدة دروس لاحظنا ظهور رسالة خطأ في احد البرامج عند اغلاقه و السبب انه هو ان أي نافذة في لغة Liberty Basic تحتاج الى برنامج فرعي تتوجه اليه عند اغلاقها ويكون هذا البرنامج الفرعي غالبا لاغلاق مقابض handles الملفات المفتوحة ضمن هذه النافذة و المقابض handles للنافذة المفتوحة نفسها و ثم انهاء البرنامج بالعبارة end .وهذا البرنامج هو البرنامج السابق ولكن هذه المرة يعمل بشكل صحيح:

```
open "textwindow" for text as #1
print #1,"!trapclose [quit]"
open "d:\test.txt" for input as #2
print #1,"!contents #2"
wait

[quit]
close #2
close #1
end
```

يوجد عدة اختيارات يمكن ان تكتبها في صيغة اظهار نافذة النصوص و هذه الاختيارات تحدد تصرف او سلوك نافذة النصوص.

فتح نافذة بصورة طبيعية

```
open window title for text as #handle
```

فتح نافذة على ابعاد الشاشة full-screen

```
open window title for text_fs as #handle
```

فتح نافذة بدون scroll bars أي ماتسمى اشرطة التمرير الموجودة اسفل و بجانب النافذة.

```
open window title for text_nsb as #handle
```

فتح نافذة بلا اشرطة تمرير و مع صندوق لكتابة النصوص من نوع inset .

```
open window title for text_nsb_ins as #handle
```

نافذة الرسوم graphics :

يمكنك من خلال هذه النافذة رسم مجموعة اشكال و كذلك وضع و تحريك الـsprites و هي الاشكال التي تتكون منها الالعاب ثنائية البعد.اما اذا اردت الرسم داخل نافذة اخرى و هذه النافذة ليست graphics يمكن ذلك عن طريق عنصر اخر هو الـ graphic box ,ولكن غالبا ما تصمم الالعاب في عنصر graphic box و هذا العنصر يكون ايضا في graphic window .وفي هذا الدرس شرح لكيفية و طرق الرسم بلغة Liberty Basic .

ان كل شاشة تتكون من نقاط صغيرة تسمى pixels و من خلال هذه النقاط تتكون الصورة على الشاشة و ماهي الصورة الا مجموعة من النقاط مرتبة,و في الحاسبات

الحديثة تستطيع النقطة الواحدة ان تعرض اكثر من ستة عشر مليون لون فلذلك تصبح الصورة على شاشة الحاسوب واضحة الى درجة كبيرة جدا,وكل لون من هذه الالوان يتكون من خليط ثلاثة قيم هي اللون الاحمر و الاخضر و الازرق وهي تسمى RGB اختصارا لـRed و Green و Blue ,حيث ان اللون الناتج هو خليط من هذه الالوان بنسب مختلفة ويستطيع كل لون ان يتخذ قيمة من 0-255 .كما انه في الدروس القادمة ستلاحظ ان هناك عنصرين اساسيين في الرسم وهو الـ Foreground وهو لون خط الرسم,أي الفرشاة التي ستستعملها للرسم على الشاشة,و الـ Background و هو لون خلفية النافذة.كما ان الرسوم قد تحتاج الى اعادة الرسم بعد ان تغير حجم النافذة أي عمل redraw.كذلك ان الرسم يعتمد على حجم النقطة (حجم فرشاة الرسم) وهذه تحدد خشونة الخطوط المرسومة على الشاشة,وهناك نوع من الرسم الذي يسمى turtle graphics وتستعمل لرسم خطوط باتجاهات مختلفة بطريقة سهلة.وكذلك ان لهذه اللغة ميزة جيدة جدا وهي امكانية جمع مجموعة من الرسوم المرسومة في الـ graphic box أو الـ graphic window في مجموعات (segments) ولكل مجموعة رقم ما و يمكن حذف كل مجموعة من الرسوم عن طريق امر delsegment وذلك من خلال رقم الـ segment .كذلك يجب ملاحظة وجود حالتين لفرشاة الرسم اثناء الرسم و هي حالة down أي يمكنك من رسم أي شئ و الحالة الاخرى هي up ولا يمكنك الرسم عندما يكون up ,ويتم الرسم باختيار نقطة بداية للرسم من خلال احداثيات x و y .

يمكن ان ترسم الرسوم في الـ graphic box او الـ graphic window و اعتقد ان الرسم في الـ graphic box افضل,والـ graphic box هو عنصر من النافذة التي يوجد فيها,ولكل عنصر ضمن نافذة مقبض أي handle تمكننا من التعامل مع هذا العنصر و هذه المعلومة تنطبق على كل عناصر هذه اللغة ,فمثلا اذا كان صندوق الرسم(graphic box)يقع ضمن نافذة الرسم (graphic window) وكان مقبض نافذة الرسوم هو w# مثلا ,فسيكون مقبض صندوق الرسوم w.ext# حيث ext تمثل أي اسم أو حرف تريد وضعه وغالبا مايختاره المبرمجون ليكون الحرف الاول للعنصر المراد وضعه لسهولة تذكره مثلا (graphic box) سيكون الحرف g مثلا,مع ملاحظة انه يجب تعريف صندوق الرسم أي تحديد موقعه من النافذة و ابعاده أيضا,مع ملاحظة انه كل عناصر النافذة مهما كانت يجب تعريفها قبل فتح النافذة التي تحتويها.

فتح نافذة الرسوم يكون بالصيغة التالية:

```
open window title for graphics as #handle
```

وله اشكال اخرى ايضا كما في الـ text window :

```
open window title for graphics_fs as #handle
 open window title for graphics_nsb as #handle
```

ويمكن وضع صفتين أو أكثر على ان تفصل بالرمز (_)

```
open window title for graphics_fs_nsb as #handle
open window title for graphics_nf_nsb as #handle
```

حيث nf تشير الى انه ليس للنافذة اطار ولايمكن تغيير ابعادها بواسطة المستخدم.

تحضير الـ graphic box:

الصيغة

```
graphicbox #handle.ext,x,y,width,height
```

مثال

```
graphicbox #w.g,10,10,300,300
open "window" for graphics as #w
print #w,"trapclose [quit]"
wait
```

[quit]
close #w
end
لاحظ اننا قد استعملنا جملة trapclose التي تؤدي الى استدعاء البرنامج الموجود في الـ label الموجود في صيغتها و هو [quit] حيث يجب استعمال هذا الامر مع كل النوافذ.لكن مع نافذة الـ text قمنا بوضع الرمز(!) أمامه وذلك لتجنب طبع الجملة trapclose في صندوق النصوص,لأن لغة Liberty Basic ستعتبره نصا عاديا تريد طبعه في النافذة لذلك يجب وضع الرمز (!) قبل كل الاوامر الموجهة الى نافذة النصوص text window .كذلك لاحظ في البرنامج اعلاه ان للـ graphic box و الـ graphic window نفس الـhandle الرئيسي وهو w# لأننا نريد أن يكون صندوق الرسم ضمن نافذة الرسوم.
ملاحظة:بعض اوامر الرسم قد لاتعمل مع صندوق الرسم.

اوامر الـ graphic box:

print #handle.ext,"setfocus"
نقل التركيز الى صندوق الرسوم فسيصبح قادرا على استقبال الاوامر وخاصة اوامر الضغط على مفاتيح لوحة المفاتيح عندما يكون التركيز focus على صندوق الرسوم.

print #handle.ext,"enable"
جعل صندوق الرسوم فعالا active ليتمكن من استقبال الاوامر.

print #handle.ext,"disable"
جعل صندوق الرسوم جامدا أي لايستقبل الاحداث مثل حركة مؤشر الفأرة و لوحة المفاتيح.

print #handle.ext,"show"
جعل صندوق الرسوم ظاهرا و ليس مخفيا.

print #handle.ext,"hide"
اخفاء صندوق الرسوم.

Print #handle,"autoresize"
أي تغيير الابعاد التلقائي فاذا غيرت ابعاد النافذة التي يتواجد فيها صندوق الرسم سيغير صندوق الرسم ابعاده لتتناسب مع بعد حافته عن حافة النافذة.

الاوامر التالية قد تنطبق على صندوق الرسم او قد لاتنطبق و سوف يتم التنبيه عن ذلك في حال ان الامر لايمكن تطبيقه على صندوق الرسم أو نافذة الرسم.

Print #handle,"backcolor color"
تغيير لون خلفية النافذة او صندوق الرسوم,و color يمثل اللون مثلا اخضر green وهذا اللون سيكون اللون الثانوي أي لون تلوين الاشكال المختلفة وغيرها ويمكن ان يكون من الالوان التالية:
black, blue, brown, buttonface, cyan, darkblue, darkcyan, darkgray, darkgreen, darkpink, darkred, green, lightgray, palegray, pink, red, white, yellow.

كما يمكن كتابة ثلاثة ارقام مفصولة عن بعضها بفراغ واحد و ترتيبها من اليسار الى اليمين red و green و blue وكل منها يأخذ قيمة من 0-255 :
print #w,”backcolor 0 255 0”

print #handle,"box x y"
رسم شكل رباعي باللون المحدد في جملة color (سيتم شرحه بعد عدة اوامر) و يتم رسمه عن طريق تحديد النقطة x,y و التي تحدد احدى زوايا الشكل الرباعي أما النقطة المقابلة لها فهي موقع مؤشر الرسم (و الذي سيتم شرح كيفية تحديده بعد عدة اوامر),فيتم من خلال هاتين النقطتين معرفة عرض و ارتفاع الشكل الرباعي.

print #handle,"boxfilled x y"
يتم رسم شكل رباعي ملون باللون المحدد في جملة backcolor ,أما الجملة السابقة فهي لرسم شكل رباعي بدون تلوين أي رسم اطار فقط.

print #handle,"circle r"
يتم رسم دائرة بتحديد نصف قطر الدائرة,حيث r هي قيمة نصف قطر الدائرة أما نقطة المركز فهي موقع مؤشر الرسم حاليا.

print #handle,"circlefilled r"
رسم دائرة ملونة باللون المحدد في جملة backcolor .

print #handle,"cls"
مسح النافذة أو صندوق الرسوم من كل الرسوم.

print #handle,"color c"
ويحدد لون خط الرسم و c يمثل الالوان التي شرحناها في الجملة backcolor .

print #handle,"ellipse w h"
رسم شكل بيضوي مركزه مؤشر الرسم و احد قطريه w و الاخر h .

print #handle,"ellipsefilled w h"
رسم شكل بيضوي ملون.

print #handle,"fill c"
تلوين النافذة أو صندوق النصوص باللون المحدد في c حيث ممكن ان يكون قيمة لون أي اسم اللون أو قيمة RGB .

print #handle,"font facename pointsize"
تحديد خط الكتابة (رسم الكتابة) و حجم الكتابة على نافذة الرسم أو صندوق الرسم.

print #handle,"\text"
لرسم الكتابة text حيث سترسم الكتابة في الموقع الذي يمثل الزاوية السفلى اليسرى من مؤشر الرسم فمثلا لكتابة الكلمة text1 على سطر و كتابة text2 على السطر اسفل السطر الاول يجب كتابة:
print #w,”\text1\text2”
ويمكن استخدام الرمز | عندما تريد طبع الرمز \

print #w.g,"|\test1"

print #handle,"down"
تهيئة مؤشر الرسم للكتابة فأي امر يوجه الى المؤشر سيبدأ بالرسم.

Print #handle,"home"
وضع مؤشر الرسم في مركز نافذة الرسم أو صندوق الرسم.

print #handle,"up"
جعل مؤشر الرسم في وضعية عدم الرسم.

print #handle,"go A"
سيؤدي الى تحريك مؤشر الرسم عدد من النقاط pixels محددة في المتغير A ولو كان المؤشر في وضعية down سيؤدي الى رسم خط بالاتجاه الحالي لمؤشر الرسم.
Print #w,"home;down;go 60"
يمكن وضع عدد من الاوامر في نفس الجملة مع مراعاة فصلها بالرمز (;).

print #handle,"goto x y"
سيؤدي الى انتقال المؤشر الى الموقع x,y وسيرسم خط مستقيم من الموقع الحالي للمؤشر الى الموقع x,y اذا كان المؤشر في وضعية down.

print #handle,"place x y"
نقل المؤشر الى الموقع x,y من دون رسم خط مستقيم حتى لو كان المؤشر في وضعية down .

print #handle,"posxy xvar yvar"
هذا الامر يقوم بوضع قيمة x الحالية لمؤشر الرسم أي موقع مؤشر الرسم في المتغير xvar أو أي متغير عددي,و يقوم بوضع قيمة الـ y للمؤشر في المتغير yvar أو أي متغير عددي.

print #handle,"set x y"
رسم نقطة في الموقع المحدد فيx و y .

print #handle,"size a"
تحديد حجم نقطة مؤشر الرسم حيث تتأثر خشونة الخطوط المرسومة من خلال تغيير قيمة a في هذه الصيغة.

print #handle,"turn a"
تحديد اتجاه مؤشر الرسم حيث a تمثل زاوية دوران مؤشر الرسم(فرشاة الرسم).

print #handle,"flush"
بعد رسم عدة رسوم قد تريد ان تضعها في مجموعة تقوم بتسميتها بأي اسم لكي تسهل عملية حذف هذه الرسوم أو اعادة رسمها فيمكن ان تستعمل هذا الامر الذي يضع كل الرسوم السابقة و التي لم يعمل flush لها في مجموعة تحمل رقما و يمتد من رقم 1 لأول عملية flush الى أي عدد من اوامر الـ flush ,حيث ان flush الثانية تعطي رقم المجموعة 2 و هكذا..

print #w,”home;down;turn 90;go 60”
print #w,”flush”
print #w,”place 10 10;go 60”
print #w,”flush”

كما يمكن اعطاء المجموعة اسم ما بدلا من استعمال الترقيم الذاتي الذي يقوم به Liberty Basic حسب عدد مرات flush .بالصيغة التالية:

print #w,”flush segmentname”

حيث segmentname هو اسم المجموعة.

Print #handle,”delsegment segmentname”

حيث delsegment اختصار لـ delete segment و يتبعها اسم المجموعة segment name وهذا الامر يقوم بحذف المجموعة التي اسمها محدد في segment name . ولكن لن تحذف المجموعة مباشرة ولكن يجب عمل redraw للنافذة أو صندوق الرسوم لتنفيذ التغييرات.

او يمكن ان تستخدم الصيغة التالية:

print #handle,”delsegment n”

حيث n هو رقم المجموعة segment number .

print #handle,"discard"

حذف كل الرسوم التي رسمت منذ اخر عملية flush ,ولكن لن تحذف مباشرة لذلك يجب عمل redraw للنافذة أو صندوق الرسوم.

print #handle,"redraw"

عمل redraw أي اعادة الرسم لكل الرسوم في النافذة لملاحظة التغييرات وتنفيذ هذه التغييرات كحذف بعض الرسوم.كذلك توجد صيغة اخرى لهذا الامر:

print #handle,"redraw";n

حيث n تمثل رقم المجموعة segment number المراد حذفها.وهناك صيغة اخرى لهذا الامر:

print #handle,”redraw”;segmentname

حيث يجب كتابة اسم المجموعة segment .

print #handle,"segment variablename"

سيؤدي هذا الامر الى ان وضع رقم المجموعة التي ترسم حاليا في المتغير variablename ,ويمكن الحصول على اخر مجموعة تم عمل flush لها من خلال هذه القيمة –١ أي variablename-1 .

print #handle,"line x1 y1 x2 y2"

رسم خط مستقيم بين النقطة x1,y1 و النقطة x2,y2 .

print #handle,"locate x y width height"

هذا الامر لـ graphicbox فقط وليس للـ graphic window ويقوم باعادة تعيين موقع و ابعاد الـ graphicbox في النافذة.

يمكن استدعاء صورة من النوع BMP من ملف ما عن طريق الامر loadbmp حيث نقوم باعطاء الصورة اسم خاص بلغة Liberty Basic ليتم التعامل مع الصورة بشكل أسهل:

loadbmp name,path

حيث name هو اسم الصورة المراد استخدامه من قبل هذه اللغة اثناء البرمجة,و الـ path يمثل مسار و اسم ملف صورة BMP مع الامتداد bmp .مثال على ذلك:

load "drawing1","d:\picture.bmp"

ثم يمكن رسم هذه الصورة أي وضعها في نافذة او صندوق الرسم من خلال الامر drawbmp:

print #handle,"drawbmp bmpname x y"

حيث bmpname تمثل اسم الصورة المحدد في name في الصيغة السابقة لـ loadbmp و x و y تمثل النقطة الاصل origin point للصورة و هي غالبا ماتكون في كل لغات البرمجة النهاية العليا اليسرى من الصورة وليس مركز الصورة.

عند رسم مجموعة من الرسوم قد ترغب في حفظ هذه الرسوم في ملف ما,فالخطوة الاولى هي استنساخ ذلك الجزء من الصورة من خلال الامر getbmp و الصيغة هي:

print #handle,"getbmp bmpname x y width height "

حيث x,y تمثل النقطة العليا اليسرى من المكان المراد استنساخه و الـ width يمثل عرض المكان المراد استنساخه ابتداءا من النقطة x,y و الـ height يمثل ارتفاع المكان المراد استنساخه ابتداءا من النقطة x,y ,وسيضع اسم لهذا الجزء المستنسخ و هذا الاسم محدد في bmpname في الصيغة اعلاه.وبعد استنساخ هذا الجزء من الصورة قد تريد ان تخزنه في ملف من نوع BMP. فيمكن ذلك من خلال الأمر bmpsave حسب الصيغة التالية:

bmpsave bmpname,filename

حيث bmpname هو الاسم الذي اعطيته للصورة المستنسخة في جملة getbmp و filename هو مسار و اسم الملف مع الامتداد bmp. الذي تريد خزن الصورة فيه.

ويمكن الحصول على المقبض (handle) التابع لأي صورة BMP من خلال الامر hbmp و القيمة العائدة من هذا الامر هي رقم يمثل مقبض صورة الـBMP ويستخدم حسب الصيغة التالية:

variable=hbmp("name of BMP")

حيث name of BMP هو الاسم المستخدم من قبل لغة Liberty Basic للصورة BMP و المحدد في جملة loadbmp .

أما حذف صورة الـ BMP فيتم من خلال الامر unloadbmp و الصيغة هي:

unloadbmp "name"

حيث name هو اسم الصورة (ليس اسم الملف بل الاسم المستعمل من خلال Liberty Basic).

print #handle,"pie w h angle1 angle2"

يقوم برسم شكل pie داخل شكل بيضوي ellipse الذي احد قطريه هو w و القطر الاخر h و الـ pie سيبدأ عند الزاوية angle1 و يمشي باتجاه عقارب الساعة بالزاوية angle2 اذا كانت angle2 موجبة أو عكس عقارب الساعة اذا كانت angle2 قيمة سالبة.

print #handle,"piefilled w h angle1 angle2"

تلوين الـpie بلون الـ backcolor بعد رسمه.

طباعة الرسوم على الطابعة:

يمكن طباعة الرسوم على الطابعة عن طريق توجيه الامر:

print #handle,"print"

سيتم طباعة الشكل بابعاده الحقيقية على الطابعة وذلك بارسال الامر dump كما ناقشنا في درس طباعة النصوص على الطابعة.

ويوجد اختيارات و اضافات لهذه الجملة:

print #handle,"print vga"
عندما يكون عرض الصورة المراد طباعتها يساوي 600 pixel سيتم تغيير قياسه ليساوي عرض ورقة الطابعة عند الطباعة.
print #handle,"svga"
عندما يكون عرض الصورة المراد طباعتها يساوي 800 pixel سيتم تغيير قياسه ليساوي عرض ورقة الطابعة عند الطباعة.

print #handle,"xga"

عندما يكون عرض الصورة المراد طباعتها يساوي 1024 pixel سيتم تغيير قياسه ليساوي عرض ورقة الطابعة عند الطباعة.

حيث ان هذه المسافات تمثل مقياس الرسم حيث ان كل 1024 pixel من الرسم ستساوي عرض ورقة الطابعة عند الطباعة.

قبل رسم كتابة ما على الشاشة ربما تريد معرفة عرض الكتابة بوحدة الـ pixel عند استخدام font ما,وبحجم ما, فيمكن ذلك من خلال:
print #handle,"stringwidth? vartomeasure$ width"
حيث الـvartomeasure$ تمثل النص المراد معرفة الـwidth له بوحدة الـ pixel ويكون متغير مقطعي,أما الـ width فهو متغير عددي تخزن فيه القيمة التي تمثل عرض النص.

```
open "window" for graphics as #w
print #w,"trapclose [quit]"
N$="Liberty basic"
Print #w,"font times_new_roman 15"
Print #w,"stringwidth? N$ w"
Print w
wait

[quit]
close #w
end
```

وهناك امر يتحكم بظهور او عدم ظهور شريط التمرير الافقي:
print #handle,"horizscrollbar on/off [min max]"
يمكن الغاء ظهور شريط التمرير الافقي:
print #handle,"horizscrollbar off"
و اظهار شريط التمرير:
print #handle,"horizscrollbar on"
ويمكن التحكم باكثر مدى و اقل مدى range لازاحة شريط التمرير الافقي حيث كلما زادت قيمة الـ max كلما اصبح يمكن ان تكون نافذة الرسوم اكبر.

```
open "window" for graphics as #w
print #w,"trapclose [quit]"
print #w,"horizscrollbar on 0 300"
wait
```

```
[quit]
close #w
end
```
اذا قمت بابدال العدد 300 في البرنامج السابق بعدد اكبر من 300 ستلاحظ ان المساحة المخصصة للرسم تصبح اكبر,وتستطيع ملاحظة ذلك من خلال ازاحة شريط التمرير الافقي.

وهناك امر اخر يتعامل مع شريط التمرير العمودي و له نفس الصيغة لأمر التحكم بشريط التمرير الافقي:
```
print #handle,"vertscrollbar on\off [min max]"
```

ويمكن تعديل اتجاه فرشاة الرسم الى الزاوية 270 من خلال الامر:
```
print #handle,"north"
```

بعد ان انتهينا من موضوع الرسم باستخدام البرمجة سنشرح كيفية جعل أي برنامج يتفاعل مع حركة مؤشر الفأرة مثلا و ضغط المفاتيح من لوحة المفاتيح.
يجب ملاحظة ان هذه الاحداث أي تحريك مؤشر الفأرة و ضغط مفتاح من لوحة المفاتيح تكون خاصة بعنصر تحكم ما حسب المقبض handle الذي تستعمله في الجملة,أي اذا كان المقبض خاص للنافذة فالنافذة ستصبح فعالة active لاستقبال هذه الاحداث,و يجب ان يكون التركيز focus على العنصر الذي نريد ان نستقبل الاحداث من خلاله,وحسب ماموجود في ملف الـhelp المرفق مع هذه اللغة,فان الـ graphic box لايستقبل هذه الاحداث بصورة صحيحة الا اذا وضع في نافذة من نوع Window أي نافذة عادية و سنكمل انواع النوافذ بعد هذا الدرس,أما الـsprites أي الاشكال المستخدمة في العاب الحاسبة فيمكن وضعها في الـ graphic window أو الـ graphic box .
سنشرح اولا كيفية استقبال الاحداث من لوحة المفاتيح:
يكون استقبال هذه الاحداث من خلال الامر :
```
print #handle,"when characterInput [label]"
```
هذا الامر يراقب أي ضغط للمفاتيح من خلال لوحة المفاتيح و اذا ضغط المستخدم أي مفتاح سيتوجه الى الـ label الموجود ضمن هذه الصيغة.
حيث يجب ان يحتوي هذا الـ label على اوامر و جمل تمكن البرنامج من تمييز أي المفاتيح تم ضغطها,ويتم ذلك من خلال اختبار قيمة المتغير من نوع global و المدمج مع هذه اللغة وهذا المتغير هو Inkey$ (انتبه الى حالة الاحرف الكبيرة و الصغيرة في هذه الاوامر و المتغيرات العامة المدمجة built-in مع هذه اللغة).
حيث يتم الاختبار كما يأتي:
```
if Inkey$="a" then
. . .
. . .
end if
```
أي اذا كان المفتاح المضغوط a فينفذ بعض الاوامر.وهناك اختلاف فاذا ضغطت المفتاح a وكانت الحروف في طور الحروف الكبيرة أي كان مفتاح الـ caps lock مضغوطا فلن يستجيب لهذا لأن الحرف a ليس A بالنسبة لهذا المتغير.
أما لاختبار المفاتيح الاخرى غير الحروف فيتم ذلك من خلال معرفة رمز الـ ASCII لها ,أي كما يأتي:

```
if Inkey$=chr$(32) then
. . .
end if
```

حيث ان كل مفتاح له رمز ASCII .

أما أحداث الفأرة فهي و كما هي موجودة في ملف الـhelp المرفق مع هذه اللغة:

```
leftButtonDown      - the left mouse button is now down
leftButtonUp        - the left mouse button has been released
leftButtonMove      - the mouse moved while the left button was down
leftButtonDouble    - the left mouse button has been double-clicked
rightButtonDown     - the right mouse button is now down
rightButtonUp       - the right mouse button has been released

rightButtonMove     - the mouse moved while the right button was down
rightButtonDouble   - the right mouse button has been double-clicked
middleButtonDown    - the middle mouse button is now down
middleButtonUp      - the middle mouse button has been released
middleButtonMove    - the mouse moved while the middle button was
                      down.
middleButtonDouble  - the middle mouse button has been double-
                      clicked.
mouseMove           - the mouse moved when no button was down
```

وتتم ايضا كما يأتي:

```
open "Paint something!" for graphics as #w
print #w, "when leftButtonMove [paint]"
print #w, "when characterInput [letter]"
print #w, "down ; size 3"
wait
[paint]
print #w, "set "; MouseX; " "; MouseY
wait
[letter]
print #w, "\"; Inkey$
wait
```

هذا برنامج رسم بسيط حيث يمكنك الرسم بضغط الزر الايسر للفأرة و سحب المؤشر و كذلك يمكنك الكتابة بالضغط على مفاتيح لوحة المفاتيح,وهذا مثال على كيفية استقبال الاوامر من لوحة المفاتيح و من الفأرة.وفي البرنامج اعلاه يقوم برسم نقطة في موقع مؤشر الفأرة ويقوم بهذا بكل سهولة نتيجة لوجود متغيرين من النوع global المدمجة مع اللغة و هما MouseX ويحمل قيمة الاحداثي x لمؤشر الفأرة و MouseY ويحمل قيمة الاحداثي y لمؤشر الفأرة.و المتغير Inkey$ يقوم بخزن الحرف المضغوط وهنا يقوم برسم الحرف على النافذة نتيجة وجود الرمز \ كما درست في بداية موضوع الرسوم.

By:gameprogrammer

النوافذ من نوع Window:

هي نوافذ بسيطة يمكن ان تستخدمها في مختلف البرامج ويتم تكوينها باستخدام الصيغة التالية:

```
open window title for window as #handle
```

ويمكن ان تجعل هذه النافذة بلا شريط عنوان titlebar :

```
open window title for window_popup as #handle
```

ويمكن ان تجعل النافذة بلا اطار:

```
open window title for window_nf as #handle
```

مثال:انشاء صندوق رسوم داخل نافذة من نوع window:

```
graphicbox #w.g,10,10,300,300
open "test program" for window as #w
print #w,"trapclose [quit]"
wait

[quit]
close #w
end
```

يمكن عمل update للتغييرات في هذه النافذة عن طريق print #handle,"refresh"

النوافذ من نوع dialog:

هذه النوافذ تشكل اغلب نوافذ windows وتسمى نوافذ الحوار dialog لأن المستخدم يقوم بضبط صفات البرامج من خلال هذه النوافذ لأن لها صفات تساعدها على هذا. ويمكن تكوين نافذة من هذا النوع بنفس طريقة تكوين النوافذ الاخرى لكن مع ابدال الكلمة بعد for بالكلمة dialog .

```
open window title for dialog as #w
```

فتح نافذة مساعدة:

```
open window title for dialog_modal as #w
```

وبالاضافة الى هذه الصفة فلهذه النوافذ جميع الصفات الاخرى مثل nf,fs,popup .

تستطيع منع نافذة الـ Main Window من الظهور من خلال الامر nomainwin .
يمكن التحكم بصفات النافذة ولكن قبل تكوينها من خلال الصفتين WindowHeight و WindowWidth و UpperLeftX و UpperLeftY كما في المثال التالي:

```
UpperLeftX=10
UpperLeftY=10
WindowHeight=400
WindowWidth=400
graphicbox #w.g,10,10,300,300
open "test program" for window as #w
print #w,"trapclose [quit]"
wait

[quit]
close #w
end
```

حيث UpperLeftX و UpperLeftY يمثلان الموقع x,y للنهاية العليا اليسرى من النافذة ,أما WindowHeight فيمثل ارتفاع النافذة ,و WindowWidth يمثل عرض النافذة.

هناك متغير من نوع global مدمج مع هذه اللغة هو $Platform وهو يخزن اسم نظام التشغيل أي اذا كان نظام التشغيل الذي تستعمله هو Microsoft Windows فسوف يحمل هذا المتغير المقطع الحرفي Windows ويمكن معرفة مايحويه هذا المتغير من خلال :

```
print Platform$
```

وهناك متغير من نوع global مدمج مع هذه اللغة تستطيع من خلاله معرفة نسخة لغة Liberty Basic التي تستعملها حاليا و التي يعمل عليها البرنامج الذي تكتبه حاليا وهذا المتغير هو $Version :

```
print Version$
```

هناك متغيران من نوع global يحملان ارتفاع و عرض شاشة العرض التي تستعملها حاليا هما,DisplayWidth يحمل هذا المتغير قيمة تمثل عرض شاشة العرض,و المتغير DisplayHeight يحمل هذا المتغير قيمة تمثل ارتفاع شاشة العرض,و القيم العائدة هي بوحدة الـ pixel.

```
print DisplayWidth
print DisplayHeight
```

امر تشغيل الملفات الصوتية من نوع WAV.:
يمكن تشغيل الملف الصوتي باستخدام الامر playwave و الصيغة هي:

```
playwave filename,mode
```

حيث filename هو مسار الملف الصوتي مع الامتداد extension والذي يكون هنا wav.
أما الـmode فيمثل كيفية التعامل مع الصوت وهذه امثلة على ذلك:
الانتظار الى ان ينتهي تشغيل الملف الصوتي ثم تنفيذ الجمل التي بعد هذه الجملة:

```
playwave "d:\test.wav",sync
```

عدم الانتظار الى ان ينتهي تشغيل الملف الصوتي:

```
playwave "d:\test.wav",async
```

تشغيل الملف الصوتي كل مرة حتى بعد انتهاء مدة الملف أي اعادة التشغيل كل مرة ينتهي فيها الملف الصوتي من التشغيل:

```
playwave "d:\test.wav",loop
```

ايقاف الملف الصوتي:

```
playwave ""
```

هناك نوع اخر من الملفات الموسيقية التي تسمى MIDI و التي تتميز بصغر حجم ملفاتها و امتدادها يكون midi. ,ويمكن تشغيل مثل هذه الملفات عن طريق الامر playmidi و الصيغة هي:

```
playmidi filename,length
```

حيث الـfilename يمثل الملف المراد تشغيله وعند تنفيذ هذا الامر سيخزن البرنامج طول الملف الصوتي في المتغير العددي length أو أي متغير عددي ,كما يمكنك التأكد كل فترة زمنية من ان الملف الصوتي قد انتهى لتتمكن من ايقاف الموسيقى من خلال امر يسمى ()midipos حيث يحمل هذا الامر الموقع الحالي للملف الصوتي و عندما

midipos() يساوي القيمة length فهذا يعني ان الملف الصوتي انتهى.يتم ايقاف الموسيقى عن طريق الامر stopmidi .
و البرنامج التالي يحتاج الى ملف من نوع midi. موجود في الـ Drive D: و اسم هذا الملف test :

```
'the playmidi command returns the length of the midi in
   ' the variable howLong
   playmidi "d:\test.midi", howLong
   timer 1000, [checkPlay]
   wait

 [checkPlay]
   if howLong = midipos( ) then [musicEnded]
   wait

 [musicEnded]
   stopmidi
   timer 0
   wait
```

تصميم و اظهار النوافذ:
اصبحت كل لغات البرمجة الحديثة تستطيع اظهار النوافذ و العناصر المختلفة للنوافذ كازرار التحكم و صناديق النصوص و الـ check boxex و غيرها,و تسمى هذه العناصر عناصر التحكم لأنها تجعل المستخدم قادرا على التحكم في مسار البرنامج و طبعا فان كل تغيير في هذه العناصر ينتج عنه حدث event ما و هذا الحدث يكون عبارة عن برنامج صغير ضمن البرنامج الرئيسي له label ما لنستطيع التوجه الى هذا البرنامج.فمثلا سيعرف نظام التشغيل كيف عليه أن يتصرف في حال ضغط المستخدم احد ازرار التحكم في النافذة ,وذلك يتم عن طريق استدعاء البرنامج الصغير الذي يمتلك label ما.و يجب ان يتم تعريف كل العناصر التي تريد وضعها في النافذة قبل فتح النافذة بالجملة open ,ويجب ان تمتلك مقابض من النوع #handle.ext حيث handle هو مقبض النافذة,اما الـ ext يقوم بتعريف العنصر ضمن النافذة و غالبا ما يقوم به المبرمجون هو ان يجعلوا ext عبارة عن الاحرف الاولى من عنصر التحكم ليسهل تذكر العنصر.
و النوافذ التي تستعمل هنا هي من النوع window أو dialog .

عناصر التحكم:

button

وهي الازرار التي تلاحظها في اغلب نوافذ النظام Windows .ويتم تعريفها بالصيغة التالية:

```
button #handle.ext,"title",EventHandler,corner,x,y,width,height
```

حيث title هو العنوان الذي سيظهر على الزر أي العنوان الذي يرشدك الى وظيفة الزر Button.و EventHandler هو أما label نريد التوجه اليه عندما يضغط الزر أو subroutine نريد التوجه اليه عند ضغط الزر,corner هي الزاوية التي يكون عندها نقطة الاصل origin حيث قيمة الـx و الـ y ستحسب من تلك الزاوية و يحتمل ان تكون:

UL (upper left)
UR (upper right)

LL (lower left)
LR (lower right)
الـ x,y هو موقع الزاوية corner بالنسبة للنافذة,width هو العرض و الـheight هو الارتفاع (أي ابعاد عنصر التحكم Button).
أما الاوامر التي يمكن توجيهها لهذا العنصر هي:
print #handle.ext,"string"
لتغيير عنوان الـButton حيث يكون عنوانه النص الموجود بين علامتي الاقتباس في هذه الصيغة.

print #handle.ext,"!setfocus"
التركيز focus على الـButton .

print #handle.ext,"!locate x y width height"
اعادة تعيين موقع العنصر من النافذة و اعادة تغيير ابعاده.

print #handle.ext,"!font facename pointsize"
تغيير الخط المستخدم في كتابة عنوان العنصر Button.

print #handle.ext,"!enable"
جعل العنصر فعالا Active أي يستقبل الاوامر المختلفة.

print #handle.ext,"!disable"
جعل العنصر غير فعال أي لايستقبل الاوامر.

print #handle.ext,"!hide"
اخفاء العنصر من النافذة (لكن لايحذف).

print #handle.ext,"!show"
اظهار العنصر بعد اخفاءه.

Bmpbutton:

هو زر مثل Button لكن يحوي على صورة ما محددة في صيغة تعريف هذا العنصر.
Bmpbutton #handle.ext,file,EventHandler,corner,x,y
حيث الـfile هو اسم ملف الصورة من نوع BMP و التي تريد ان تظهر على الـButton ,الـEventHandler هو اما label او subroutine و الـcorner هي الزاوية التي تحدد نقطة

الاصل origin point و الـx,y هو الموقع.ولايوجد تحديد للابعاد لأن حجم الـButton سيصبح بحجم الصورة المراد ان تظهر عليه.
الاوامر الموجهة الى عنصر التحكم:
هذه الاوامر تستطيع توجيهها حتى بعد فتح النافذة التي تحتوي هذه العناصر و تستخدم لتغيير خصائص هذه العناصر.

Print #handle.ext,"bitmap bitmapname"
تغيير الصورة على العنصر حيث bitmapname تمثل اسم الصورة و ليس اسم الملف و تستطيع استدعاء هذه الصور عن طريق الامر loadbmp .
الاوامر المتبقية هي:
print #handle.ext,"locate x y width height"
print #handle.ext,"setfocus"
print #handle.ext,"enable"
print #handle.ext,"disable"
print #handle.ext,"show"
print #handle.ext,"hide"
وقد تم توضيحها قبل هذا الدرس.

Checkbox:

هذا العنصر موجود في اغلب البرامج و هو عبارة عن مربع صغير بجانبه كتابة ما تصف هذا الاختيار فاذا قمت بتأشير المربع بالنقر عليه بالزر الايسر للفأرة فهذا يسمى set أي جعل الاختيار فعالا,اما اذا ضغطت مرة اخرى عليه فسيزول التأشير reset .
checkbox #handle.ext,"title",SetHandler,ResetHandler,x,y,width,height
الـSetHandler يمثل أما label أو subroutine تريد للبرنامج ان يتوجه اليه عندما يقوم المستخدم بتأشير صندوق الاختيار checkbox, و اما الـResetHandler فهو اما label او subroutine تريد للبرنامج أن يتوجه اليه عندما يزال التأشير من صندوق الاختيار.
الـtitle يمثل النص المراد ظهوره بجانب مربع الاختيار أي وصف هذا الاختيار للمستخدم.
الاوامر الموجهة:
print #handle.ext,"set"
تأشير مربع الاختيار.

print #handle.ext,"reset"
ازالة التأشير من مربع الاختيار.

print #handle.ext,"value? Result$"
حيث result$ متغير مقطعي ستحمل أما قيمة set اذا كان المربع مؤشرا,أو قيمة reset اذا كان المربع غير مؤشر.وهذا جيد عندما تريد التحقق من ان احدى صناديق الاختيار مؤشر أم لا.

print #handle.ext,"setfocus"
print #handle.ext,"locate x y width height"
print #handle.ext,"font facename pointsize"
print #handle.ext,"enable"
print #handle.ext,"disable"
print #handle.ext,"show"

print #handle.ext,"hide"

Radiobutton:

يشبه checkbox لكن دائما لايمكنك الا اختيار radiobutton واحد من بين أي عدد من radiobuttons .

radiobutton #handle.ext,"title",SetHandler,ResetHandler,x,y,width,height

لاتحتاج الصيغة للشرح لأنها مشابهة لصيغة checkbox .

الاوامر الموجهة:

print #handle.ext,"set"
print #handle.ext,"reset"
print #handle.ext,"value? Result$"
print #handle.ext,"setfocus"
print #handle.ext,"locate x y width height"
print #handle.ext,"font facename pointsize"
print #handle.ext,"enable"
print #handle.ext,"disable"
print #handle.ext,"hide"
print #handle.ext,"show"

groupbox:

ليس للـgroupbox أهمية كبير فتقتصر وظيفته على الاهتمام بترتيب النافذة أي وضع مجموعة من العناصر في groupbox واحد و اعطاء عنوان لهذا العنصر وذلك لتوضيح غرض العناصر الموضوعة في الـgroupbox للمستخدم.

Groupbox #handle.ext,"title",x,y,width,height

الاوامر الموجهة:

print #handle.ext,"a string"

قد ترغب في تغيير عنوان title الـgroupbox فيمكنك ذلك من خلال كتابة العنوان في مكان a string في الصيغة أعلاه.

print #handle.ext,"!locate x y width height"
print #handle.ext,"!font facename pointsize"
print #handle.ext,"!enable"
print #handle.ext,"!disable"
print #handle.ext,"!show"
print #handle.ext,"!hide"

statictext:

ويمثل مقطع من النصوص المراد ان تظهر ضمن النافذة وقد تستخدمها لتشرح للمستخدم كيفية التعامل مع البرنامج أي انه ليس لهذا العنصر أي احداث events .

statictext #handle.ext,"title",x,y,width,height

ويمكن ان تستخدمه في نافذة الـgraphics :

statictext #handle,"title",x,y,width,height

الاوامر الموجهة:

يمكن تحديد النص المراد ظهوره في الـstatictext

print #handle.ext,"a string"

print #handle.ext,"!locate x y width height"
print #handle.ext,"!font facename pointsize"
print #handle.ext,"!enable"
print #handle.ext,"!disable"
print #handle.ext,"!hide"
print #handle.ext,"!show"

Menu:

تلاحظ في اغلب البرامج وجود قوائم يمكن ان تختار منها بعض الاختيارات ,يمكن تكوين مثل هذه النوافذ في هذه اللغة:

menu #handle,"title","item1",[branchlabel1],"item2",[branchlabel2]

حيث title هو عنوان القائمة و item1 هو عنوان الاختيار الاول و [branchlabel1] هو الـlabel ألمراد التوجه اليه عند النقر بالزر الايسر للفأرة على الاختيار item1 من القائمة و كذلك الحال مع item2 و يمكن اضافة اعداد كبيرة من الاختيارات الى القوائم.كما يمكن فصل بعض الاجزاء من القائمة عن الاجزاء الاخرى بخط مستقيم و ذلك باستخدام الرمز (|):

menu #w,"File","New",[new],|,"save",[save]

الرمز (&) يمكن ان تضعه في عناوين النافذة او الاختيارات حيث ان الحرف الذي يأتي بعده سيكون الحرف الذي اذا ضغطت المفتاح ALT و ذلك المفتاح الذي يقع الرمز (&) قبله سيؤدي الى اختيار ذلك الاختيار بدلا من التوجه اليه بمؤشر الفأرة واختياره.

Popupmenu:

هي نوع من انواع القوائم التي لاتكون في اعلى النافذة و لكن يتم عرضها بالنقر على احد ازرار الفأرة في أي مكان من النافذة.

Popupmenu "item1",[branchlabel1],"item2",[branchlabel2]

ويمكن استخدام الرموز & و | كما في القوائم العادية.

حيث item1 هو عنوان الاختيار الاول الذي عندما يختاره المستخدم سينفذ البرنامج الاوامر و الجمل الموجودة في الـlabel [branchlabel1] وهكذا .

ويمكن وضع هذه الجملة في أي مكان من البرنامج و غالبا ماتوضع في حدث when RightButtonDown حيث يتم التوجه الى label ما لتكوين هذه النافذة.

Textbox:

هو صندوق لكتابة النصوص و لكن يسمح بكتابة سطر واحد فقط من البيانات و استعمال اوامره مشابه تماما لاوامر نافذة النصوص text window .

textbox #handle.ext,x,y,width,height

حيث x,y هي الزاوية العليا اليسرى من عنصر التحكم textbox .

الاوامر الموجهة:

طباعة نص ما داخل هذا العنصر.

print #handle.ext,"a string"

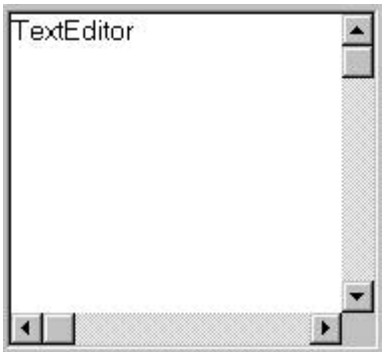
استنساخ محتويات صندوق النصوص الى متغير مقطعي.

Print #handle.ext,"!contents? Varname$"

الاوامر الاخرى:

print #handle.ext,"!locate x y width height"
print #handle.ext,"!font facename pointsize"
print #handle.ext,"!enable"
print #handle.ext,"!disable"
print #handle.ext,"!hide"
print #handle.ext,"!show"

texteditor:



هو صندوق نصوص أيضا لكن يدعم صفة تعدد السطور و يشبه نافذة النصوص text window .

texteditor #handle.ext,x,y,width,height

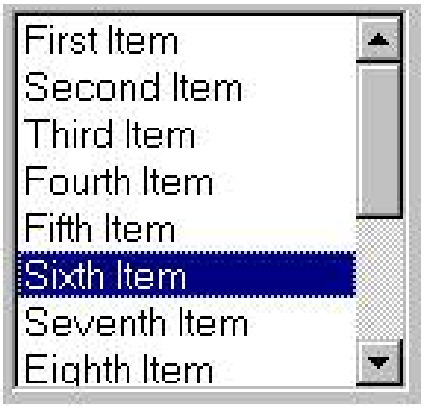
الاوامر الموجهة متشابهة تماما مع التعامل مع نافذة النصوص text window ما عدا أحد الاوامر:

هذا الامر يؤدي الى تغيير أبعاد الـtexteditor عند تغيير أبعاد النافذة التي تحتوي هذا العنصر.

Print #handle.ext,"!autoresize"

وهذه الصفة صحيحة فقط مع الـtexteditor و ليس مع الـtextbox .

listbox:



عنصر تحكم يحوي على بعض الاختيارات items و التي يستمدها من مصفوفة مقطعية فيجب ملئ المصفوفة بالعناصر المراد ان تظهر في القائمة.

Listbox #handle.ext,array$(),EventHandler,x,y,width,height

حيث array$() هو اسم المصفوفة التي يستمد منها البيانات و الـEventHandler هو label يتوجه اليه البرنامج عندما يختار المستخدم عنصرا ما من القائمة.

الاوامر الموجهة:

print #handle.ext,"select string"

يقوم باختيار العنصر المسمى string من القائمة و يمكن جعل هذا العنصر عبارة عن متغير:

print #handle.ext,"select ";string$

print #handle.ext,"selectindex I"

يقوم باختيار العنصر الذي رقم الـindex له يساوي المتغير العدد I ,حيث ان اول عنصر في القائمة صفة index له هي 1 و العنصر الثاني هو 2 و هكذا.

print #handle.ext,"selection? Selected$"

سيحمل المتغير Selected$ النص المقطعي الذي اختاره المستخدم.

print #handle.ext,"selectionindex? index"

سيحمل المتغير index القيمة التي تشير الى رقم الـindex للعنصر المختار.

Print #handle.ext,"reload"

اعادة تحميل عناصر المصفوفة داخل عنصر التحكم listbox في حالة حصل تغيير في عناصر المصفوفة.

print #handle.ext,"singleclickselect"

هذه الجملة تؤدي الى التوجه الى الـlabel الموجود في صيغة تكوين listbox و الخاصة بحدث اختيار العنصر وهو EventHandler حسب صيغة تكوين listbox ,عندما يضغط المستخدم الزر الايسر للفأرة مرة واحدة بدل من الحالة الافتراضية و التي هي double click أي النقر المزدوج للزر الايسر للفأرة لاختيار العنصر المراد من القائمة listbox .

combobox:

وهو عنصر تحكم يشبه عنصر التحكم listbox و له استعمالات كثيرة منها مثلا الاختيار من خلال مجموعة من عناصر مصفوفة مقطعية و مثلا تؤدي الى ظهور معلومات عن ذلك العنصر في texteditor مثلا وهذا يستخدم عن التعامل مع قواعد البيانات من خلال دمج التعامل مع الملفات مع التعامل مع النوافذ.

Combobox #handle.ext,array$(),EventHandler,x,y,width,height

ويستمد العناصر من مصفوفة مقطعية.

الاوامر الموجهة:

print #handle.ext,"!string"

يقوم بتغيير العنصر المختار حاليا من قائمة الـ combobox الى النص المقطعي الذي يأتي بعد الرمز (!) في الصيغة أعلاه.

Print #handle.ext,"contents? string$"

المتغير $string سيحمل النص المختار حاليا من هذا العنصر.

أما بقية الاوامر فهي متشابهة مع الـlistbox .

في بعض الاحيان قد تريد ان تتحكم ببعض صفات هذه العناصر ولهذا تستعمل الـstylebits .

stylebits #handle.ext,addbits,removebits,extendedaddbits,extendedremovebits

حيث الـhandle.ext# تمثل المقبض (handle) للعنصر المراد تغيير صفاته.
واذا اردت اضافة صفة فتضيفها الى المكان addbits في الصيغة اعلاه و اذا اردت اضافة اكثر من صفة فيجب ان تكون مفصولة عن بعضها البعض بالكلمة or, أما الـremovebits هي الصفات المراد حذفها من النافذة.و أما البقية فهي للنوافذ التي استخدمت الـextendedstyles في تكوينها و اعتقد انه لو وضعنا القيمة 0 فهو افضل لأن النوافذ التي ننشئها ليست من النوع extendedstyle .

WINDOW AND CONTROL STYLE CONSTANTS

window styles - some also work for controls:

_WS_BORDER Creates a window that has a thin-line border.

_WS_CAPTION Creates a window that has a title bar (includes the WS_BORDER style).

_WS_HSCROLL Creates a window that has a horizontal scroll bar.

_WS_MAXIMIZE Creates a window that is initially maximized.

_WS_MAXIMIZEBOX Creates a window that has a Maximize button.

_WS_MINIMIZE Creates a window that is initially minimized. Same as the WS_ICONIC style.

_WS_MINIMIZEBOX Creates a window that has a Minimize button.

_WS_VSCROLL Creates a window that has a vertical scroll bar.

button styles:

_BS_LEFT Left-justifies the text in the button rectangle.

_BS_RIGHT Right-justifies text in the button rectangle.

_BS_RIGHTBUTTON Positions a radio button's circle or a check box's square on the right side of the button rectangle.

editbox (textbox) styles:
_ES_CENTER Centers text in a multiline edit control.

_ES_PASSWORD Displays an asterisk (*) for each character typed into the edit control.
_ES_RIGHT Right-aligns text in a multiline edit control.

listbox styles:
_LBS_MULTICOLUMN Specifies a multicolumn list box that is scrolled horizontally.
_LBS_SORT Sorts strings in the list box alphabetically.

statictext styles:
_SS_CENTER Specifies a simple rectangle and centers the text in the rectangle.
_SS_RIGHT Specifies a simple rectangle and right-aligns the given text in the rectangle.

وهذا برنامج على هذا الامر (البرنامج موجود في ملف الـhelp المرفق مع هذه اللغة):

```
stylebits #main.pw, _ES_PASSWORD, _ES_AUTOVSCROLL or
_ES_MULTILINE, 0, 0
textbox #main.pw, 10, 10, 250, 25

'here is one right justified, and we use a handle variable
justHandle$ = "#main.rjust"
stylebits #justHandle$, _ES_RIGHT, 0, 0, 0
textbox #main.rjust, 10, 40, 250, 25

'here's an example of twiddling style bits for a window
stylebits #main, _WS_SYSMENU, _WS_POPUP, _WS_EX_CONTEXTHELP, 0
open "STYLEBITS demo" for window_popup as #main

#main.pw "please"
#main.rjust "on the right"
wait
```

يمكن اختيار لون خلفية النافذة عن طريق:

```
BackgroundColor$=color$
```

حيث $color متغير يمثل اللون

```
BackgroundColor$=”green”
```

و يمكن اختيار لون الـforeground

```
ForegroundColor$=color$
```

اذا تمت كتابة هاتين الجملتين قبل فتح النافذة المراد تغيير الوانها سيكون التغيير بالالوان فقط للنافذة ,أما اذا كتبت هاتين الجملتين بعد فتح النافذة فان لون خلفية كل

عناصر التحكم و النافذة سيكون نفس اللون الموجود في $BackgroundColor و لون الكتابة ستكون نفس لون الكتابة الموجودة في $ForegroundColor .
كذلك يمكن ان تغير الوان بعض عناصر التحكم:

TextboxColor$
ComboboxColor$
ListboxColor$
TexteditorColor$

قد تحتاج في برامجك الى بعض صناديق الحوار dialog boxes مثل اختيار لون ما أو اختيار ملف ما,و لاختصار ذلك توجد مجموعة كبيرة من صناديق الحوار المدمجة مع هذه اللغة:

color dialog:

يتيح اختيار لون من نافذة تعرض الالوان:

colordialog color$,chosen$

حيث $color متغير مقطعي يدل على اللون المختار حاليا وقد يكون قيمة RGB أما اللون الذي ستختاره فستخزن قيمته في المتغير $chosen .

confirm:

يستعمل للسؤال عن شئ ما و يجيب المستخدم بـyes أو no

confirm string;var$

حيث string هو السؤال المراد طرحه على المستخدم و اختيار المستخدم سيخزن في المتغير $var .

filedialog:

يستخدم لاختيار الملفات.

Filedialog title$,templatestring$,var$

حيث $title سيكون عنوان نافذة اختيار الملف,و $templatestring متغير مقطعي يشير الى أي نوع من الملفات تريد عرضه في النافذة مثلا "*.txt" أي كل الملفات بالامتداد txt. و $var يحمل مسار و اسم الملف المختار من النافذة.و يمكنك استخدام أي متغيرات مقطعية مع هذه الصيغ أي ليس من المفروض ان تستعمل المتغيرات المذكورة في الصيغة أعلاه.

Fontdialog:

لاختيار نوع الخط و صفات الخط.

Fontdialog fontspec,fontspecvar$

حيث fontspec هو اسم الـfont و صفاته مثل "corier_new 10 italic" و الخط المختار سيخزن مع صفاته في المتغير $fontspecvar .

notice:

لعرض رسالة ما على الشاشة لتنبيه المستخدم.

Notice string$

حيث $string أما متغير مقطعي أو نص مقطعي موجود بين علامات الاقتباس و يمثل محتويات الرسالة المراد عرضها على المستخدم.

Printerdialog:

لعرض نافذة اختيار الطابعة و عدد من الصفات الاخرى كاختيار عدد النسخ المراد طبعها لعرض هذه النافذة استخدم الامر PRINTERDIALOG :

PRINTERDIALOG

ستظهر نافذة اختر منها الاختيارات التي تناسبك ,في هذه الحالة هناك ثلاثة متغيرات من نوع global ستحتفظ بالقيم التي اخترتها من النافذة هي $PrinterName وهو اسم الطابعة و PrintCollate و PrintCopies .

ولاختيار الخط المستخدم في الطباعة هناك متغير يسمى $PrinterFont يمكن ان تعطيه قيمة أسم الخط المراد و صفاته مثل:

"corier_new 10 bold"

prompt:

يمكنك من خلال هذه النافذة ان تعرض سؤال و يقوم المستخدم بالاجابة و الاجابة هنا لن تكون فقط yes و no بل يمكن كتابة نصوص مقطعية:

prompt question$;var$

حيث $question يمثل السؤال المراد طرحه على المستخدم و الاجابة التي سيكتبها المستخدم ستخزن في المتغير $var .أي ان هذه النافذة تشبه عبارة input المستخدمة في السؤال عن شئ ما.

وهناك برنامج يمكن ايجاده في ملف الـhelp المرفق مع هذه اللغة يمكن من خلاله تصميم النوافذ عن طريق الفأرة دون كتابة سطر واحد من تصميم النوافذ و هذا البرنامج يقوم بتكوين برنامج يمكنك استنساخه الى محرر هذه اللغة,و هذا البرنامج سيساعد في تكوين النوافذ و عناصرها ويسمى هذا البرنامج freeform.

التعامل مع مكتبات الربط الديناميكية Dynamic Linking Libraries (DLL):

ان نظام Windows بكافة انواعه يتميز بواجهة رسومية Graphical User Interface ,أي وجود ازرار و صناديق نصوص و غيرها,و ان البرنامج الذي تشغله و الذي يحوي هذه العناصر يعمل بسرعة و نظام Windows لايتأثر عند تشغيل عدة نوافذ تستخدم هذه العناصر.و الفكرة هنا انه لو كانت كل نافذة تحوي مقطع من البرنامج ليقوم برسم كل عنصر من هذه العناصر ,وربما كانت هناك عدة نوافذ ,فذلك يؤدي الى وقف نظام Windows عن العمل بسبب استهلاك موارد النظام System Resources و احتواء الذاكرة المؤقتة Ram على الكثير من المعلومات نتيجة الرسم (رسم العناصر على كل نافذة).لذلك ولحل هذه المشكلة ظهرت ملفات الـDLL وهي اختصار لـ Dynamic Linking Library ,وكل ملف من هذه الملفات يحتوي على دوال Functions قد يختلف عددها و تختلف و ظائفها من ملف لاخر,و هذه الدوال هي اجزاء من البرامج تمت كتابتها سابقا و هي مشفرة بطريقة ما,فمنها ما يستخدم لرسم العناصر و منها مايستخدم للتحكم بالنظام التشغيلي Windows كتغيير العناوين التي تظهر في النوافذ و غيرها,ولهذا فعند كتابة برنامج ما,وهذا البرنامج يستخدم واجهة رسومية Graphical Interface وبعد عمل Compiling له ,سيقوم الـCompiler بوضع عناوين و مسارات ملفات الـDLL التي استخدمنا الدوال التابعة لها في البرنامج,لذلك يجب ان تكون ملفات الـDLL موجودة و لا تحذف أي توزع مع البرنامج الرئيسي,و في نظام Windows توجد مجموعة من هذه الملفات في المسار c:\windows\system .

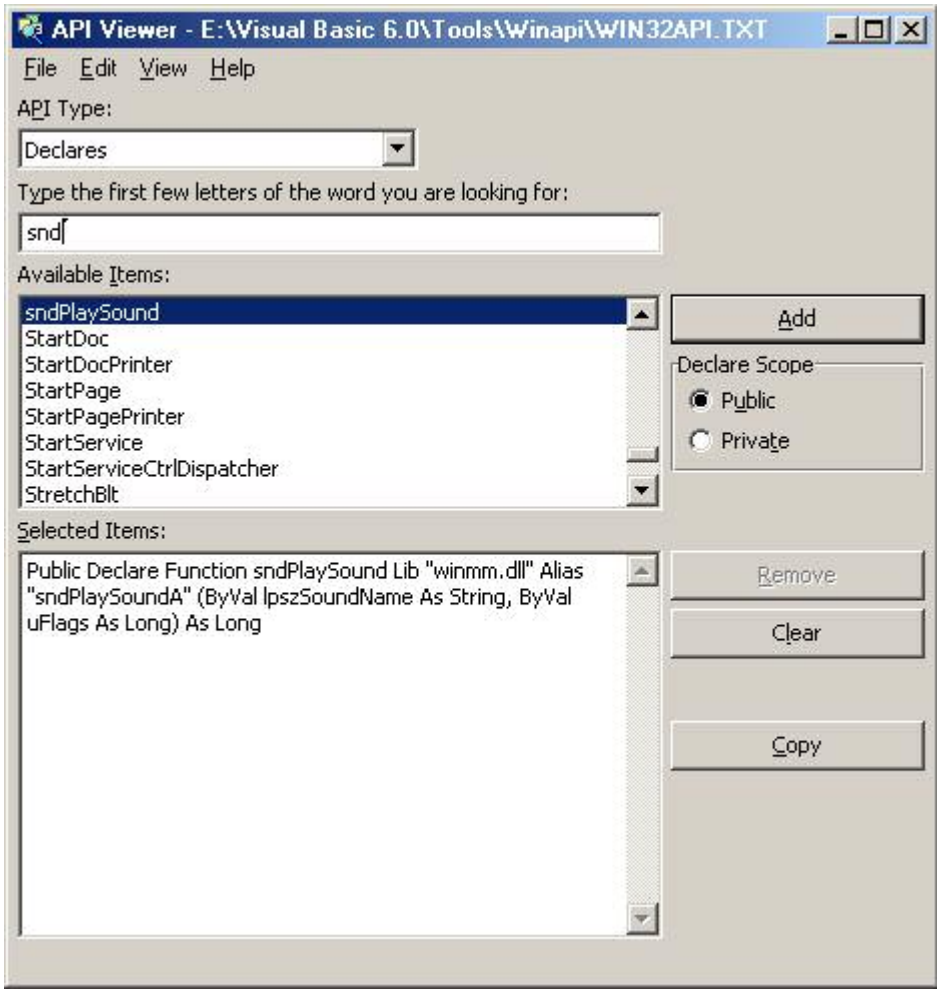
ان هذه الدوال تحتاج الى parameters (وهي القيم التي تريد تمريرها للدالة) وهذه الـparameters تحتاجها الدالة لاكمال المهمة المطلوبة ,مثلا هناك دوال معينة لتلوين الشاشة ,الـparameter الرئيسي هنا سيكون بالتأكيد متغير ما يمثل اللون الذي نريد تلوين الشاشة به.اضافة الى ذلك فكل دالة API (وكل عملية تحصل في الحاسوب) تعود منها قيمة ما تسمى القيمة العائدة returned value و هي قد تكون عبارة عن عدد ما قد يكون 1 أو 0 أو أي عدد اخر ,مثلا هناك دالة هي GetPixel وهي تستخدم لمعرفة لون نقطة معينة على الشاشة و تحتاج الى عدة متغيرات و لكن الاهم من هذا ان القيمة العائدة منها هي لون ما,و هذا اللون يمثل لون النقطة في احداثي ما على الشاشة.اضافة الى هذا ,ان المتغيرات التي نمررها للدالة قد تكون من نوع Integer أي اعداد صحيحة أو Long أي اعداد كبيرة (اعتقد انها اعداد اقل أو اكبر من 32000 و اعتقد انها اكثر من 16000000).أو قد تكون الـparameters هي عبارة عن نصوص مقطعية (strings) وتمثل في لغة Liberty Basic بـ(ptr).وهذه الـptr و الـlong و الـinteger و غيرها تسمى الـtypes أي انها انواع البيانات المراد تمريرها الى الدالة.و هذه قائمة بالـtypes التي يمكن استخدامها مع لغة Liberty basic .

TYPES

double	(a double floating point)
ulong	(4 bytes, unsigned long integer)
long	(4 bytes, signed long integer)
short	(2 bytes, signed short integer)
ushort, word	(2 bytes, unsigned short integer)
ptr	(4 bytes, long pointer, for passing strings)
struct	(4 bytes, long pointer, for passing structs)
void, none	(a return type only, used when a function doesn't return a value)

boolean	(true/false expression)

وهذه تستخدم مع الجملة struct و التي سنشرحها لاحقا.
في القائمة اعلاه تلاحظ بعض المصطلحات مثل signed و unsigned حيث ان هذه هي انواع من الاعداد,حيث لو اخذنا المتغير integer و افترضنا انه يستطيع استقبال القيم من ٣٢٠٠٠ الى –٣٢٠٠٠ (ففي هذه الحالة هذا المتغير هو من نوع unsigned integer)أي integer يحمل اشارة أما موجبة أو سالبة,أما الـsigned integer ففي هذه الحالة ستمتد قيمته من 0 الى 64000 أي الاعداد بالاشارة الموجبة فقط.
هناك دوال معينة قد لاترجع لنا قيمة ما,و في هذه الحالة سيكون المتغير الذي سيستقبل القيمة العائدة هو من نوع void .أما المتغيرات من نوع الـboolean تتقبل قيمتين أما true أو false ,حيث true يشير الى تحقق الدالة و false يشير الى عدم تحقق الدالة.
كيفية الحصول على هذه الدوال؟
هذه الدوال ليست قليلة العدد لهذا يوجد برنامج مرفق مع مجموعة برامج visual studio (ويشمل visual c++ و visual basic وغيرها) وهذا البرنامج يسمى API text viewer .حيث يمكن كتابة اسم الدالة و الحصول على مكونات الدالة و ملف الـDLL الذي تنتمي اليه.



الشكل اعلاه هو لبرنامج API text viewer ,وتجد ملف مرفق مع هذا البرنامج هو WIN32API.TXT حيث يستمد هذا البرنامج هذه الدوال من هذا الملف,ويمكن ان تختار هذا الملف من قائمة File ثم Open ,ويمكن كتابة الحروف الاولى للدالة التي تريد استخدامها في الحقل الاول ويمكن اختيار الدالة من الحقل الثاني Available Items و سيظهر الـDeclaration أو التصريح الخاص بهذه الدالة,وهذا التصريح يستخدم مع لغة Visual Basic فقط و سنتعلم كيفية تحويله ليلائم لغة Liberty Basic لاحقا.
ويمكن ان نشرح مكونات هذا التصريح وهو تابع للدالة sndplaysound وهي دالة لتشغيل صوت ما:

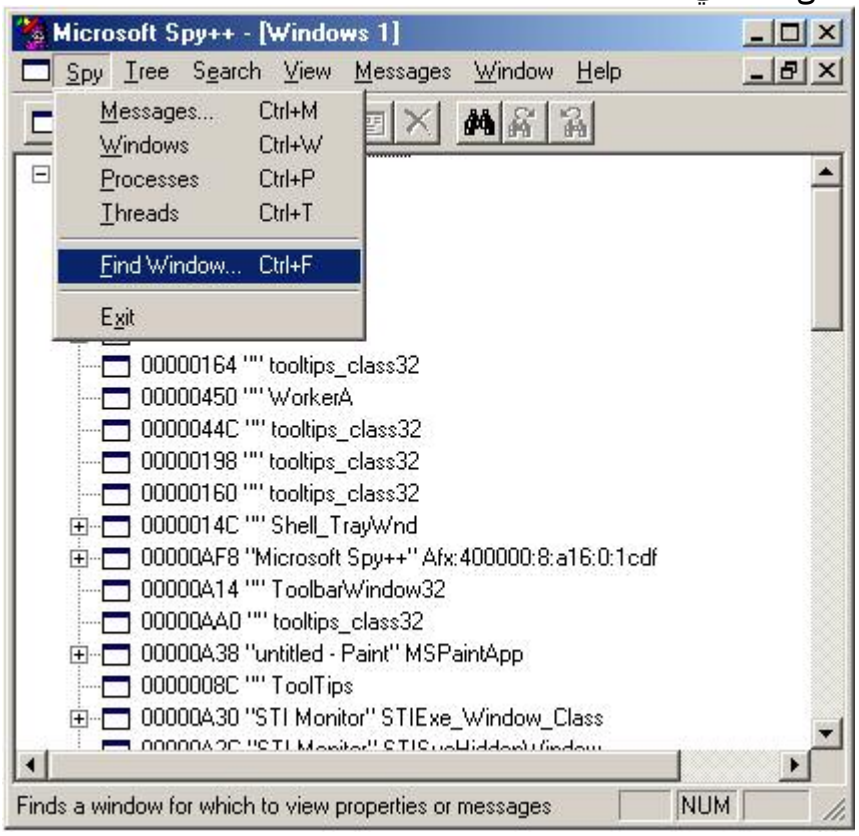
Public Declare Function sndPlaySound Lib "**winmm.dll**" Alias "**sndPlaySoundA**" (**ByVal lpszSoundName As String, ByVal uFlags As Long**) ***As Long***

ان ما يهمنا من هذا التصريح لنستعمله مع لغة Liberty Basic هو النص الذي تحته خط فقط في الصيغة اعلاه,فالنص الاول winmm.dll يشير الى ان الملف الذي تستمد منه هذه الدالة هو الملف winmm.dll و الموجود في مجلد (folder) الذي يحمل الاسم system و الموجود ضمن المجلد windows .اما النص الثاني و هو sndPlaySoundA فيشير الى اسم الدالة التي تستدعيها و يجب مراعاة استخدام الحروف الكبيرة أو الصغيرة الموجودة في هذه الصيغة فهي case-sensitive اما النص الثالث وهو

ByVal lpszSoundName As String, ByVal uFlags As Long)

فكلمة ByVal تعني نوع تمرير المتغيرات و قد درست ذلك في موضوع الـFunctions و معنى byval هو passing by value .

اما **lpszSoundName** فيمثل المتغير الاول في الدالة و هو من النوع string أي يتقبل نصوص أو متغيرات مقطعية فقط,و هذا واضح من الجملة **As String** اما المتغير الاخر فيمكن معرفته بنفس الطريقة,و هناك جملة ***As Long*** في نهاية التصريح و هو يمثل نوعية القيمة العائدة من استعمال هذه الدالة.
ان الجملة As string أو As long يمكن ان تتغير من دالة الى اخرى فقد تكون As void و الكلمات بعد As ثابتة فقد تكون:
ulong,long,short,ushort,word,ptr,string,void,boolean
أما غير هذه الكلمات اذا جاءت بعد As فتكون انواع اخرى من المتغيرات تسمى structures وسيتم شرحها لاحقا,اما اذا جاء بعد كلمة As الكلمة const فهذا معناه ان هناك constant ما,و الـconstant هو ثابت يستعمله نظام Windows نفسه لانجاز وظائف معينة و في نهاية هذا الدرس هناك موقع يشرح جميع دوال الـAPI وفي هذا الموقع تجد شرحا لكل الـconstants للدالة التي تريدها,و ليس لكل الدوال constants أو structures ,بل بعضها له هذه الصفة فقط,و غالبا ماتكون الدوال التي تقوم بأعمال معقدة هي التي تحتاج هذه الانواع من المتغيرات.
أما اذا جاء بعد الكلمة As الكلمة hwnd فهذا يعني ان الدالة تحتاج الى الـhandle أو المقبض للنافذة أو غيره.اما ما هي المقابض فهي ببساطة ارقام معينة يستخدمها النظام Windows لمعرفة و تمييز النوافذ و اقصد بالنوافذ انها ليست فقط نوافذ النظام Windows بل حتى كل اداة داخل النافذة تمتلك رقم handle خاص بها.اما اذا وجدت hdc فهذا يدل على الـdevice context و تحتاج بعض الدوال الى هذا المتغير و يتم استخراج الـhdc من الـhwnd لأي نافذة عن طريق الدالة GetDC .
و يمكن معرفة hwnd لكل نافذة في الـwindows من خلال معرفة الـclass الخاص بها حيث يتم معرفة hwnd لأي نافذة من خلال الدالة FindWindow و التي تحتاج الى اسم الـclass و يتم الحصول على اسم الـclass من خلال برنامج Spy++ المرفق مع Visual studio ويستعمل كالتالي:



حيث يتم اختيار find window من القائمة Spy فتظهر النافذة التالية:

لاحظ الشكل الذي يشبه المروحة و المحاط بالمربع باللون الاحمر للتوضيح,يمكن سحب هذا الشكل الى أي نافذة عن طريق الفأرة و بمجرد وصول الشكل الى النافذة ستظهر معلومات عن النافذة في النافذة اعلاه و هذا المعلومات هي caption أي النص الموجود في النافذة و Class وهو اسم أو رقم يمثل نوع النافذة (النافذة قد تكون أي عنصر من عناصر التحكم فربما TextBox و ربما ComboBox و غيرها).
وهناك خطوات لاستخدام ملفات الـDLL :

١.فتح ملف الـDLL .
٢.استدعاء الدالة المطلوبة.
٣.غلق ملف الـDLL .

فتح ملف الـDLL:

```
open “filename.dll” for dll as #handle
```

يتم استدعاء الدالة من الملف بالصيغة التالية:

```
calldll #handle,”function”,parameter1 as type,parameter2 as type,r as result
```

حيث الـhandle هي نفس الـhandle التي فتح بها ملف الـDLL ,و function هو اسم الدالة و يجب كتابته بالشكل الصحيح مع مراعاة وجود احرف كبيرة و صغيرة أي انه case-sensitive. و parameter1 هو احد متغيرات (parameters) الدالة و الـtype هو نوعه فقد يكون نص مقطعي أو عدد من النوع long ,ويلي هذا متغيرات الدالة كلها,وعندما تنتهي من كتابة متغيرات الدالة يجب ان يكون هناك متغير ليستقبل القيمة العائدة من الدالة و هذا المتغير هو r في الصيغة اعلاه ونوعه قد يختلف وغالبا مايكون عدد من نوع long .

ثم غلق ملف الـDLL :

close #handle

يوجد طريقة لاستدعاء الـDLL الخاصة بنظام Windows بدون فتح الملف و لاباغلاقه اذ تضع لها Liberty Basic مقابض(handles) خاصة بها وهذه الـhandles هي:

#user32
#kernel32
#gdi32
#winmm
#shell32
#comdlg32
#comctl32

حيث تستخدم كمقابض لدوال الـAPI .
حيث ان هناك ملفات في نظام Windows بالامتداد DLL. وكل ملف يحتوي عدة دوال
حيث اذا تواجدت دالة ما في الملف user32.exe فيجب استعمال الـhandle التي تسمى #user32 .

للحصول على قيمة مقبض (handle) للنافذة يجب استخدام الامر hwnd وصيغة هذا الامر كالتالي:

h=hwnd(#handle)

هذه الدالة تخزن قيمتها العائدة في المتغير h و القيمة العائدة هي مقبض (handle) النافذة بالنسبة للنظام Windows, أما handle الموجودة بين القوسين في الصيغة اعلاه هي التي تعطى للنافذة في جملة open .

انشاء الـstructures:
في بعض دوال الـAPI قد تحتاج الى بعض الـstructures فيمكن تعريفها بالصيغة التالية:

struct name,field1 as type,.....

واذا اردت وضع قيم معينه لاحدى fields للـstructure يمكن ذلك من خلال الصيغة التالية:

name.field1.struct=value

حيث value تساوي القيمة المراد وضعها في الحقل field1 من الـstructure .

هذه بعض الامثلة التي تمثل انواع المختلفة من دوال الـAPI وهي تمثل كيفية استخدام الـ constants و الـstructures أي قد لاتفهم هذه الدوال و لكن ستعرف كيفية تمرير المتغيرات و الثوابت في الدوال المختلفة و اذا لم تفهم عمل هذه الدوال فيمكن ان تدرس الدوال المستعملة في البرامج هنا عن طريق الموقع www.allapi.net واختيار API List من الجانب الايسر من الصفحة الرئيسية للموقع ,فيمكن بعد ذلك اختيار أي دالة من الدوال المرتبة حسب الحروف الابجدية.
وهذه الامثلة موجودة في ملف الـhelp المرفق مع لغة Liberty Basic .

```
'CALL32-4.BAS  -  Make some API calls to play wave files and

'dynamically resize a window
```

```
open "kernel32" for dll as #kernel
open "user32" for dll as #user
open "winmm" for dll as #mm
open "Me" for window as #aWindow

print str$(playMode)

wavefile$ = "chimes.wav"
playMode = 4
calldll #mm, "sndPlaySoundA", _
    wavefile$ as ptr,_
    playMode as long,_
    result as long

hndl = hwnd(#aWindow)

for x = 50 to 350 step 5

    calldll #user, "MoveWindow", _
        hndl as ulong, _
        50 as long, _
        50 as long, _
        x as long, _
```

```
        x as long, _
        1 as long, _
        result as boolean

    next x

    input r$

    progname$ = "notepad.exe"
    code = _SW_SHOWNA

    notice str$(code)

    calldll #kernel, "WinExec", _
        progname$ as struct, _
        code as ushort, _
        result as ushort

    print result
    close #kernel
    input r$

'*********************************************
```

```
'CALL32-5.BAS  -  make various API calls to play
'wave files, track
'the mouse position, and move a window around

struct point, x as long, y as long

open "kernel32" for dll as #kernel
open "user32" for dll as #user
open "Me" for window as #aWindow

hndl = hwnd(#aWindow)

for i = 1 to 500

    calldll #user, "GetCursorPos", _
        point as struct, _
        result as void

    x = point.x.struct
    y = point.y.struct

    calldll #user, "MoveWindow", _
```

```
        hndl as ulong, _
        x as long, _
        y as long, _
        100 as long, _
        100 as long, _
        1 as long, _
        result as boolean

next x

progname$ = "notepad.exe call32-5.bas"
code = _SW_NORMAL

notice str$(code)

calldll #kernel, "WinExec", _
    progname$ as struct, _
    code as long, _
    result as long

print result

close #kernel
```

```
    input r$

'*********************************
'WINRECT.BAS - show how to get window position and size

  'and demonstrate how to use the struct statement

   struct winRect, _
      orgX as long, _
      orgY as long, _
      crnrX as long, _
      crnrY as long

   open "test me" for window as #win

   open "user32.dll" for dll as #user

   hndl =  hwnd(#win)

   calldll #user, "GetWindowRect", _
      hndl as ulong, _
      winRect as struct, _
      result as long

   print "Upper Left x, y of 'test me': "
   print winRect.orgX.struct; ", "; winRect.orgY.struct

  print
   print "Lower Right x, y of 'test me': "
   print winRect.crnrX.struct; ", "; winRect.crnrY.struct

   close #user
   close #win
   wait

   end
```

للغة Liberty basic صفة تسمى Logical Line Extension أي امكانية تقسيم السطر الواحد من البرنامج الى عدة سطور منفصلة بالرمز(_) كما في هذا الجزء من البرنامج أعلاه:

```
struct winRect, _
      orgX as long, _
      orgY as long, _
      crnrX as long, _
      crnrY as long
```

التعامل مع الـsprites:
الـsprites هي الشخصيات الموجودة في كل الالعاب الثنائية البعد و التي تتحكم بها و يتحكم الحاسوب بقسم اخر منها و سيتم في هذا الدرس شرح كيفية التعامل مع هذه الاشكال ثم كيفية بناء لعبة ثنائية البعد في الحاسوب,أما عن جودة الالعاب فهذا يتوقف على قدرة المبرمج على البرمجة الجيدة و الاشكال التي يستخدمها في الالعاب.حيث يمكن تكوين أي لعبة ثنائية البعد و العاب ثنائية البعد تبدو و كأنها ثلاثية البعد نتيجة تغيير بعض قياسات الاشكال و الحسابات الرياضية.و لكل sprite موقع يحتله على الشاشة و يتم وضع الـsprite أما في نافذة الرسوم أو صندوق الرسوم.و ان موقع الـsprite يتمثل بالاحداثي x,y و الذي غالبا مايكون في النهاية العليا اليسرى من الشكل و يمكن جعله في مركز الشكل عن طريق أحد الاوامر.و في هذه اللغة لابد أن يكون للـsprite شئ يسمى mask و هذا يمثل صورة باللونين الابيض و الاسود فقط يمثل اللون الاسود المناطق التي تريد لهذه اللغة ان تظهرها على الشاشة و اللون الابيض يمثل المناطق الشفافة من الشكل و التي ستقطع من الشكل لكي لاتظهر حدود حول الشكل فتكون اشكال اللعبة غير جيدة.
وهذه بعض الاشكال من ملف المساعدة help المرفق مع هذه اللغة و هذه الصور موجودة مع هذه اللغة عندما تشتريها من الانترنت.

تمثل هذه الصورة شكل الـsprite وهذا الشكل لو تم وضعه كـsprite فسيظهر بمظهر غير جيد للالعاب الحديثة أي لايعقل ان تظهر حدود الشكل ايضا و هي باللون الاسود.و دائما يجب ان تكون حدود الشكل المرسوم باللون الاسود كما في الشكل اعلاه,أي ان المساحات غير المستعملة يجب ان تكون باللون الاسود.و ان الالعاب تتكون من هذه الاشكال sprites و التي تتحرك على صورة تمثل خلفية الشاشة.

الصورة السابقة غير منطقية في الالعاب.

يجب ان تظهر الالعاب كما في الصورة اعلاه..
هذا الـsprite قد تم عمل masking له ليظهر بهذا الشكل (بلا الحدود باللون الاسود).

الصورة اعلاه تظهر الصورة التي تستعمل في اظهار الشكل بلا الحدود باللون الاسود.أي ان الـmasking في هذه اللغة هو عبارة عن صورتين مدمجتين في ملف واحد(تعتبر صورة واحدة)الصورة السفلى تمثل الشكل الحقيقي و الصورة العليا تمثل الـmask و يجب ان يكون الـmask بنفس ابعاد الصورة السفلى و كل لون اسود في الصورة السفلى يحول الى لون ابيض في الصورة العليا (الـmask),وهذا دقيق جدا لذلك يوجد في ملف المساعدة لهذه اللغة برنامج يسمى mask editor يستقبل الصورة الاصلية و يقوم بعمل الـmask و تستطيع حفظ الصورة الناتجة في ملف من نوع BMP, و ان كل الصور المستعملة في هذه الاشكال يجب ان تكون من نوع BMP و يمكن استخدام صور من نوع JPG و لكن باستخدام دوال الـAPI. فالشكل المعروض على الشاشة سيكون بلا حدود باللون الاسود.ولهذه الاشكال صفات مختلفة منها صفة التصادم أي اذا تصادم شكلان من هذه الاشكال يمكن معرفة من المصطدم و توجيه البرنامج الى عمل حدث ما,ولكن كيفية تحديد الاصطدام هي تلامس المربع المحيط بالشكل الاول مع المربع المحيط بالشكل الثاني أي حتى المنطقة الشفافة التي لانراها أي ان منطقة التصادم هي المربع المحيط بالشكل في الصورة الاصلية و يمكن تجاهل تلك الحدود و حساب الاصطدام (collision) فقط بين الاشكال الظاهرة أي لاوجود للمنطقة المربعة للتصادم أي ان حدود الشكل هي حدود التصادم.
قد تكون ابعاد الـ sprite التي تستعملها صغيرة فيمكن تغيير حجم الـsprite عن طريق عمل scaling أي تغيير القياس.كما عندما تغير موقع الشكل يجب تحديث الاطار frame (لأن الحركة عبارة عن اطارات) ليأخذ الشكل موقعه الجديد.يتم تحريك الشكل بتغيير موقع الـx,y له.
يجب تحميل الصور التي ستشكل الـsprites عن طريق الامر loadbmp .

addsprite:

يستخدم لتكوين sprite :

print #handle,"addsprite spritename bmpname"

ويمكن ان يحتوي الـsprite على عدة صور متسلسلة لتكوين حركة animation :

print #handle,"addsprite spritename bmp1 bmp2 bmp3 ....."

background:

تستعمل لجعل صورة معينة كخلفية للنافذة أو صندوق الرسم.

Print #handle,"background bmpname"

Backgroundxy:

و تؤدي الى عمل scroll أي تحريك لخلفية الشاشة الى الاحداثيات الموجودة في الصيغة:

print #handle,"backgroundxy bmpname x y"

centersprite:

تؤدي الى جعل نقطة الاصل بالنسبة للـsprite الموجود في صيغتها في المركز بدلا من الجهة العليا اليسرى من الـsprite .

print #handle,"centersprite spritename"

cyclesprite:

عمل animation للـsprite فتشاهد ان صورته تتغير و هي عبارة عن الصور التي اضفتها للـsprite في جملة addsprite .

print #handle,"cyclesprite spritename 1"

عمل animation من الصورة الاولى الى الاخيرة.

print #handle,"cyclesprite spritename -1"

عمل animation من الصورة الاخيرة الى الاولى.

ويمكن اضافة كلمة once الى كل من الصيغتين ليكون عرض الـ animation مرة واحدة فقط في كل مرة تستدعي فيها هذا الامر.

print #handle,"cyclesprite spritename 1 once"

drawsprites:

لن يتم عرض أي من الـsprites الا عند استدعاء هذا الامر عند كل frame للحركة أي يمكن أن تجعله الامر الاخير في الـtimer وهذا الامر يقوم بتحديث مواقع الـsprites في الشاشة.

Print #handle,"drawsprites"

Removesprite:

لحذف الـsprite من الشاشة.

Print #handle,"removesprite spritename"

Spritecollides:

يستخدم لمعرفة اسماء الـsprites التي تصطدم مع الـsprite الموجود في صيغة هذا الامر في اللحظة الحالية,ويقوم هذا الامر بخزن اسماء جميع الـsprites المصطدمة مع الـsprite في متغير مقطعي ويفصل بين كل اسمين من اسماء الـsprites مسافة.

Print #handle,"spritecollides spritename list$"

حيث spritename هو اسم الـsprite المراد معرفة الـsprites الاخرى المصطدمة معه و التي تخزن اسماؤها في المتغير list$ حسب الصيغة أعلاه.

Spriteimage:

كما عرفنا سابقا ان الـsprite يتكون من مجموعة من الصور فيمكن ان تجعل صورة واحدة من هذه الصور هي الصورة الحالية للـsprite :

print #handle,"spriteimage spritename bmpname"

حيث bmpname هي احدى الصور المعرفة في الامر addsprite عند تعريف الـsprite الذي يسمى spritename حسب الصيغة اعلاه.

Spritemovexy:

لتحريك الـsprite عدد من الـpixels بالاتجاهين الافقي x و العمودي y .

print #handle,"spritemovexy spritename x y"

حيث x تمثل عدد الـpixels أي الخطوات التي يتحركها الـsprite بالاتجاه الافقي,و y هي للاتجاه العمودي.

Spriteoffset:

في هذا الامر سيقوم المبرمج باعطاء قيمتين لكل من x و y وتمثل هذه القيم قيم ازاحة,حيث اذا وجهت الامر الى هذه اللغة لتقوم بوضع الـsprite في الموقع 100,30 مثلا,فسيكون الموقع الناتج هو 100+x,30+y, أي ان الموقع يزاح بالمقدار المعطى في هذا الامر:

print #handle,"spriteoffset spritename x y"

spriteorient:

تغيير اتجاه الـsprite وهناك 4 اتجاهات :

normal,flip,mirror,rotate 180

print #handle,"spriteorient spritename flip"

spriteround:

تفترض هذه اللغة ان حدود تصادم الشكل هي عبارة عن مربع يمثل حجم الصورة الاصلية,و لكن بتوجيه هذا الامر لهذه اللغة سيتم جعل منطقة التصادم هي حدود الشكل الظاهر.

Print #handle,"spriteround spritename"

Spritescale:

تغيير ابعاد الـsprite ,حيث يتم تغييره حسب النسبة التي تعطيها في صيغة هذا الامر:

print #handle,"spritescale spritename percent"

حيث percent هي النسبة,و يمثل هذا التغيير تغييرا في الارتفاع و العرض للـsprite .

ملاحظة:عند استخدام متغير ما في أغلب صيغ الـsprite يجب ان يستعمل خارج الصيغة,و لنفرض ان الـpercent في المثال السابق هي متغير عددي:

percent=30

print #w.g,"spritescale name ";percent

أي انه اذا كانت الصيغة تتمثل بعدد ثابت و اردنا وضع متغير فيجب ان يكون خارج الصيغة.

Spritetofront:

عند اظهار الـsprites على الشاشة وعند تحريكها ستبدو و كأن بعضها يغطي الـsprites الاخرى و يخفيها,وذلك يرجع الى الترتيب الذي تتبعه هذه اللغة في رسم الـsprites فما يرسم اولا سيكون عمقه depth في الشاشة اكثر و سيظهر تحت البقية,وان اخر sprite مرسوم سيظهر فوق البقية.و بهذا الامر تستطيع اختيار الـsprite الذي تريده أن يرسم فوق الـsprites البقية.

Print #handle,"spritetofront spritename"

Spritetoback:

يرسم الـsprite الموجود في صيغته قبل كل الـsprites و لذلك يظهر تحت الـsprites البقية.

Print #handle,"spritetoback spritename"

Spritetravelxy:

هذا الامر يسهل الكثير من الامور حيث يمكن تحريك الـsprite من مكان الى اخر عن طريق تحديد النقطة المراد الانتقال اليها,حيث يظهر الانتقال و كأن الـsprite يطير الى تلك النقطة.

Print #handle,"spritetravelxy spritename x y speed [label]"

ينتقل الـsprite المسمى spritename الى النقطة x,y بالسرعة speed و عندما يصل الى النقطة x,y سيؤدي الى تنفيذ الاوامر الموجودة في الـlabel الموجود في الصيغة.

print #w.g,"spritetravelxy name ";x;" ";"10";" ";"2";" ";"[events]"

الجملة اعلاه هي مثال على استخدام المتغيرات مع الثوابت العددية في نفس الصيغة ,لاحظ ان الثوابت العددية و غير العددية محاطة بعلامات اقتباس.وهذه العبارات تعد الاصعب بالنسبة للمبتدئين بالبرمجة.

Spritevisible:

اخفاء أو اظهار الـsprite و حتى عندما يكون الـsprite مخفيا فانه يكون خاضعا لقوانين التصادم لأنه موجود و ليس محذوف و لكنه مخفي فقط.

لاظهار الـsprite:

Print #handle,"spritevisible spritename on"

لاخفاء الـsprite:

print #handle,"spritevisible spritename off"

spritexy:

عند اضافة الـsprite يجب وضعه على مكان في الشاشة ,ويتم هذا من خلال هذا الامر وذلك بتحديد الموقع x,y :

print #handle,"spritexy spritename x y"

spritexy?:

قد تحتاج لمعرفة موقع أحد الـsprite المتحركة أو غير المتحركة في الوقت الحالي لذلك يمكنك من خزن قيم الاحداثيات التي تمثل موقعه الحالي في متغيرين من النوع العددي:

print #handle,"spritexy? spritename x y"

عند تنفيذ هذا الامر ستخزن قيم الاحداثيات x و y التي تمثل الموقع الحالي للـsprite في المتغيرين x و y في الصيغة اعلاه.

يرجع السبب الى عدم كتابة امثلة للموضوع sprites الى وجود الكثير من الامثلة عن الـsprites و المرفقة مع هذه اللغة.

هناك طريقتين لتأخير البرنامج وذلك لابطاء تنفيذ أوامر معينة :

١.الطريقة الاولى fori=1 to 1000:next I هذا السطر سيحسب من 1 الى 1000 ثم ينفذ الاوامر التي تقع بعده ويمكن زيادة الفترة بزيادة العدد 1000 و جعله 10000 مثلا.

٢.الطريقة الثانية باستخدام دوال الـAPI

calldll #kernel32, "Sleep",50 as long, re as long

حيث يمثل الرقم 50 فترة التأخير بوحدة الـmillisecond .

مثال على الـbackground scrolling :

```
loadbmp "b","d:\test.bmp"
graphicbox #w.g,10,10,300,300
open "scrolling" for window as #w
print #w,"trapclose [quit]"
print #w.g,"background b"
for I=5 to 1000 step 5
scan
calldll #kernel32, "Sleep",50 as long, re as long
print  #w.g ,"backgroundxy ";x;" 0"
print  #w.g, "drawsprites"
next I
```

مثال على تحريك sprite :

```
loadbmp "b","d:\test.bmp"
loadbmp "p1","d:\walk1.bmp"
loadbmp "p2","d:\walk2.bmp"
graphicbox #w.g,10,10,300,300
open "sprites" for window as #w
```

```
print #w,"trapclose [quit]"
print #w.g,"addsprite walk p1 p2"
print #w.g,"spritexy walk 150 150"
print #w.g,"background b"
for I=5 to 1000 step 5
scan
calldll #kernel32, "Sleep",50 as long, re as long
print #w.g,"cyclesprite walk 1"
print  #w.g ,"backgroundxy ";x;" 0"
print  #w.g, "drawsprites"
next I
```

يجب وضع الـ label الذي يسمى [quit] :

```
[quit]
close #w
end
```

يفترض هذا البرنامج وجود صورة تسمى test.bmp في الـDrive D:\ .
الامر scan يبقي البرنامج مستعدا لتلقي احداث events اخرى بدلا من الانشغال بالمهمة الحالية,وهذه الاحداث تشمل مراقبة مدخلات لوحة المفاتيح و الفأرة.
يستعمل التأخير خاصة في الحاسبات الحديثة التي تمتلك معالجات حديثة وذلك لعمل فترة تأخير لتحريك الاشكال.

تصميم الالعاب باستخدام Liberty Basic:

هذه المقالات تشرح اساسيات تصميم الالعاب بهذه اللغة لأن تصميم الالعاب متشابه في كل اللغات تقريبا و حتى تصميم الالعاب الثلاثية الابعاد يتشابه مع تصميم هذه الالعاب,و تتنوع صعوبة برمجة اللعبة باختلاف فكرة اللعبة فهناك الالعاب البسيطة جدا و هناك المعقدة و خاصة الالعاب التي تتكون من مراحل و الالعاب التي يجب ان تجعل الحاسبة تتحكم بالخصوم داخل اللعبة و كتابة برنامج يؤدي الى تصرف شخصيات اللعبة بطريقة ذكية.وكل لعبة مهما بلغت درجة تعقيدها يتحكم بها مؤقت خاص ليتحكم بحركة الشخصيات التي تتحكم بها الحاسبة.وفيما يأتي خطوات تصميم الالعاب:

١.تحميل الصور المستخدمة في اللعبة باستخدام الامر loadbmp.
٢.تكوين مجموعات الـsprites باستخدام الامر addsprite .
٣.وضع الـsprites في المواقع الابتدائية (initial positions) باستخدام الامر spritexy.

ملاحظة:من الافضل استخدام الـgraphic box بدلا من الـgraphic window .

٤.وضع جملة الـtimer لاستدعاء الـlabel الرئيسي في اللعبة.

٥.وضع جملة للتوجه الى label ما عند الضغط على مفتاح ما من لوحة المفاتيح أو تحريك الفأرة (Mouse) أو النقر على أحد ازرار الفأرة من قبل المستخدم,باستخدام الجملة when characterInput أو غيرها من جمل استقبال مدخلات لوحة المفاتيح او الفأرة.
٦.وضع جملة setfocus للمقبض handle الذي تستعمله للعبة.فمثلا اذا كنت تستخدم الـgraphicbox و كان الـhandle هو (#w.g) قستكون الجملة "print #w.g,"setfocus

٧.الـ label الرئيسي و الذي يستدعيه المؤقت timer كل فترة,لأن احداث اللعبة تحتاج الى التغيير دائما فالخصوم الموجودة في اللعبة يجب ان تتحرك كل فترة زمنية و كلما كانت الفترة الزمنية أقصر كلما كانت اللعبة اسرع و لكن يجب على الاقل ان تكون القيمة 56 ,timer 56,[main] .

٨.وضع الـlabel الذي سيتوجه اليه البرنامج عند استقبال مدخلات من لوحة المفاتيح أو الفأرة.

٩.الـlabels الاضافية التي قد تحتاجها في البرنامج.

يجب وضع الامر drawsprites في نهاية الـlabel الذي يستدعى من قبل المؤقت ليضمن الـupdate في مواقع الـsprites.

وفائدة المؤقت timer في الالعاب انه يمكن كتابة برنامج ليقوم بعمل checking لمتغيرات معينة في اللعبة او لتغيير مواقع الخصوم في اللعبة.

تصميم برنامج لتحريك الـsprite في الاتجاهات الاربعة و منعه من الخروج من حافات الشاشة.

```
Loadbmp "spr2","d:\sprite.bmp"

WindowHeight=400
WindowWidth=400

xpos=150:ypos=150

Open "sprite example" for graphics as #w
Print #w,"when characterInput [move]"
Print #w,"addsprite spr spr2"
Print #w,"spritexy spr 150 150"
print #w,"drawsprites"

Print #w,"trapclose [quit]"
[main]
```

```
Print #w,"setfocus"
scan
goto [main]

[move]

h$=Inkey$
if h$="w" then ypos=ypos-5
if h$="s" then ypos=ypos+5
if h$="d" then xpos=xpos+5
if h$="a" then xpos=xpos-5
print #w,"spritexy? spr x y"
if x>280 then xpos=280
if x<0 then xpos=0
if y>280 then ypos=280
if y<0 then ypos=0

print #w,"spritexy spr ";xpos;" ";ypos
print #w,"drawsprites"
goto [main]

[quit]
close #w
end
```

By:gameprogrammer

الامر scan ضروري جدا فهو يجعل هذه اللغة تراقب لوحة المفاتيح و الفأرة للتأكد من انها مستعملة أو غير مستعملة حاليا من قبل المستخدم.
البرنامج السابق يفترض وجود صورة من نوع BMP في الـDrive D: تسمى sprite.bmp

وبرنامج اخر يمثل شكلين sprites احدهما يتحكم به المستخدم و الاخر تتحكم به الحاسبة,واذا تصادم الشكلان يعود كل منهما الى نقطة البداية,يجب توفر صورتين من نوع BMP في الـDrive D: ,يمكنك رسم أي شكلين صغيرين ثم اجراء mask editing لهم باستعمال mask editor, و يجب ان تكون مسارات هذه الملفات كالتالي:

```
d:\sprite.bmp
d:\sprite2.bmp
```

البرنامج:

```
Loadbmp "spr2","d:\sprite.bmp"
loadbmp "spr3","d:\sprite2.bmp"

WindowHeight=400
```

```
WindowWidth=400

xpos=150:ypos=150:spr4.x=40:spr.y=40:spr4.speed=2

Open "sprite example" for graphics_nsb as #w

Print #w,"addsprite spr spr2"
Print #w,"spritexy spr 150 150"

print #w,"addsprite spr4 spr3"
print #w,"spritexy spr4 40 40"

print #w,"drawsprites"
Print #w,"trapclose [quit]"


timer 100,[timing]
Print #w,"when characterInput [move]"
Print #w,"setfocus"
wait

[timing]
gosub [computer.control]
gosub [collision.detection]
print #w,"drawsprites"
wait


[computer.control]
print #w,"spritexy? spr4 x4 y4"
if x4>=300 then spr4.speed=-2
if x4<=0 then spr4.speed=2
newx4=x4+spr4.speed
print #w,"spritexy spr4 ";newx4;" ";"40"
return

[collision.detection]
print #w,"spritecollides spr4 list$"
if instr(list$,"spr")<>0 then
print #w,"spritexy spr 150 150"
print #w,"spritexy spr4 ";spr4.x;" ";spr4.y
xpos=150
ypos=150
spr4.speed=2
end if
return
```

```
[move]
h$=Inkey$
if h$="w" then ypos=ypos-5
if h$="s" then ypos=ypos+5
if h$="d" then xpos=xpos+5
if h$="a" then xpos=xpos-5
print #w,"spritexy? spr x y"
if x>350 then xpos=350
if x<0 then xpos=0
if y>320 then ypos=320
if y<0 then ypos=0
print #w,"spritexy spr ";xpos;" ";ypos

wait


[quit]
close #w
end
```

لكي يسهل تسمية المتغيرات يمكن استعمال النقطة point من لوحة المفاتيح.فمثلا لتسمية الاحداثي x للـsprite الذي يسمى basic يمكن كتابة:

```
basic.x=20
```

وهكذا,وكذلك يمكن استعمال نفس التسمية في الـlabels .
في الخطوة القادمة سنقوم بوضع background للنافذة و نضع بعض الاشكال الاخرى كحواجز لاتسمح بالمرور من خلالها.

أما برنامج Mask Editor فيمكن الحصول عليه من ملف الـhelp وذلك بالدخول الى القسم sprites ثم من موضوع add mask يمكن اختيار قائمة Edit ثم copy ثم في محرر لغة Liberty Basic اختيار paste من قائمة edit و مسح كل محتويات الملف ماعدا البرنامج ثم حفظه في ملف ثم تنفيذه run .

البرنامج التالي عبارة عن تطوير للبرنامج السابق حيث تم اضافة خلفية الشاشة و تم اضافة مايسمى في الالعاب بالعوائق و هي الاجسام التي تقف عائق اما الاجسام الاخرى أو الشخصية الرئيسية للعبة و في هذا البرنامج عندما تحرك الشخصية الرئيسية للعبة ستصطدم بهذا العائق (water) و يمكن ان نعتبره مسطح مائي لاتستطيع شخصية اللعبة المرور من خلاله,وهذا موجود في اغلب الالعاب,حيث ستتوقف الشخصية الرئيسية في اللعبة عند الاصطدام بهذا الجسم,و ذلك بالحصول على الاحداثيات القديمة للشخصية الرئيسية ثم اختبار فيما اذا كان الجسم يصطدم مع العائق واذا كان قد اصطدم يجب ارجاع الشخصية الرئيسية الى الاحداثيات القديمة,وتؤخذ الاحداثيات القديمة قبل مقطع البرنامج المتمثل بتحريك الجسم عند الضغط على المفاتيح w للتحريك للاعلى,s للأسفل,a لليسار,d لليمين:

```
Loadbmp "spr2","d:\sprite.bmp"
loadbmp "spr3","d:\sprite2.bmp"
loadbmp "b1","d:\green.bmp"
loadbmp "water1","d:\water.bmp"
```

```
WindowHeight=400
WindowWidth=400

xpos=150:ypos=150:spr4.x=40:spr.y=40:spr4.speed=2

Open "sprite example" for graphics_nsb as #w

print #w,"background b1"

print #w,"addsprite water water1"
print #w,"spritexy water 60 100"

Print #w,"addsprite spr spr2"
Print #w,"spritexy spr 150 150"
print #w,"spriteround spr"

print #w,"addsprite spr4 spr3"
print #w,"spritexy spr4 40 40"

print #w,"drawsprites"
Print #w,"trapclose [quit]"


timer 100,[timing]
Print #w,"when characterInput [move]"
Print #w,"setfocus"
wait

[timing]
gosub [computer.control]
gosub [collision.detection]
print #w,"drawsprites"
wait


[computer.control]
print #w,"spritexy? spr4 x4 y4"
if x4>=300 then spr4.speed=-2
if x4<=0 then spr4.speed=2
newx4=x4+spr4.speed
print #w,"spritexy spr4 ";newx4;" ";"40"
return

[collision.detection]
print #w,"spritecollides spr4 list$"
if instr(list$,"spr")<>0 then
print #w,"spritexy spr 150 150"
print #w,"spritexy spr4 ";spr4.x;" ";spr4.y
```

```
xpos=150
ypos=150
spr4.speed=2
end if
return

[collision.with.water]
print #w,"spritecollides water list2$"
if instr(list2$,"spr")<>0 then
print #w,"spritexy spr ";oldx;" ";oldy
end if
return

[move]
h$=Inkey$
print #w,"spritexy? spr oldx oldy"
if h$="w" then ypos=ypos-5
if h$="s" then ypos=ypos+5
if h$="d" then xpos=xpos+5
if h$="a" then xpos=xpos-5
print #w,"spritexy? spr x y"
if x>350 then xpos=350
if x<0 then xpos=0
if y>310 then ypos=310
if y<0 then ypos=0
print #w,"spritexy spr ";xpos;" ";ypos
gosub [collision.with.water]
wait


[quit]
close #w
end
```

الالعاب من نوع Side-Scroller:

وهي الالعاب التي يمكن للشخصية الرئيسية للعبة ان تتوجه الى اليمين أو الى اليسار من الشاشة و تظهر أشياء جديدة أي ان المرحلة في اللعبة لا تقتصر على شاشة واحدة و لكن تمتد على أضعاف أبعاد الشاشة.ولتوضيح ذلك ,هذه صورة ملتقطة من لعبة Mario لجهاز الـNES المنتج من قبل شركة Nintendo :

لاحظ الخط المستقيم باللون الاخضر,هذا الخط ليس جزء من اللعبة لكنه للتوضيح,فحركة الشاشة و مكوناتها (sprites) تتم عند وصول الشخصية الرئيسية الى الخط المستقيم في الشكل اعلاه,أي ان الشاشة لن تتحرك الى ان تصل الى هذا الخط و تستمر بالسير بذلك الاتجاه ويمكن عمل ذلك عن طريق هذه اللغة عن طريق الامر backgroundxy ثم زيادة قيمة الاحداثي x عند وصول الشخصية الرئيسية للعبة الى مكان معين,اما الاشكال (sprites) لتحريكها يجب ان توضع في مكان غير مرئي من الشاشة أي انه اذا كان الجزء الظاهر من اللعبة عبارة عن مساحة بأبعاد 300*300 pixel فيمكن ان تضع الاشكال التي لا تريد ظهورها حاليا في احداثي اكثر من 300 ,ثم تغير موقعها أي تحركها الى اليسار عند وصول الشخصية الرئيسية للعبة في المنطقة بالخط المستقيم في الشكل أعلاه.
البرنامج التالي يستعمل ثلاثة sprites أحدهما يمثل الشخصية الرئيسية و المتبقيين يمثلان اشكال تقع خلف المنطقة المرئية من النافذة,ويستعمل البرنامج التالي background للنافذة لتوضيح عملية الـscrolling .

```
Loadbmp "spr2","d:\sprite.bmp"
loadbmp "spr3","d:\sprite2.bmp"
loadbmp "b1","d:\green.bmp"
loadbmp "water1","d:\water.bmp"
global c2
c2=0
WindowHeight=300
WindowWidth=300

xpos=10:ypos=100
global spr4.x
spr4.x=500
global spr.y
spr4.y=40
```

```
global water.x
water.x=400
global water.y
water.y=100


Open "sprite example" for graphics_nsb as #w

print #w,"background b1"

print #w,"addsprite water water1"
print #w,"spritexy water 400 100"

Print #w,"addsprite spr spr2"
Print #w,"spritexy spr 10 100"
print #w,"spriteround spr"

print #w,"addsprite spr4 spr3"
print #w,"spritexy spr4 500 40"

print #w,"drawsprites"
Print #w,"trapclose [quit]"
Print #w,"when characterInput [move]"

[main]
Print #w,"setfocus"
scan
goto [main]

[move]
h$=Inkey$
if h$="d" then xpos=xpos+5
if h$="a" then xpos=xpos-5
print #w,"spritexy? spr x y"
if h$="d" and x=120 then xpos=120:call scroll 2
if x<0 then xpos=0
print #w,"backgroundxy ";c2;" ";0
print #w,"spritexy spr ";xpos;" ";ypos
print #w,"drawsprites"
goto [main]



[quit]
close #w
end
```

```
sub scroll c
c2=c2+c
water.x=water.x-2
spr4.x=spr4.x-2
print #w,"spritexy water ";water.x;" ";water.y
print #w,"spritexy spr4 ";spr4.x;" ";spr4.y
end sub
```

لاحظ انه تم استخدام متغيرات من نوع global وذلك لكي تستطيع استخدامها داخل أو خارج البرنامج الفرعي subroutine, البرنامج الفرعي scroll يقوم بعملية الـscrolling للنافذة بمكوناتها وذلك بتغيير قيم الاحداثيات و اعادة وضع الاشكال في المواقع الجديدة,أما بالنسبة للـbackground فيجري تغيير احداثياتها داخل الـsubroutine ثم يمكن اعادة تعيين موقعها خارج البرنامج الفرعي لكي لاترجع الى الاحداثي 0,0 في كل مرة تضغط فيها أحد المفاتيح a أو d .

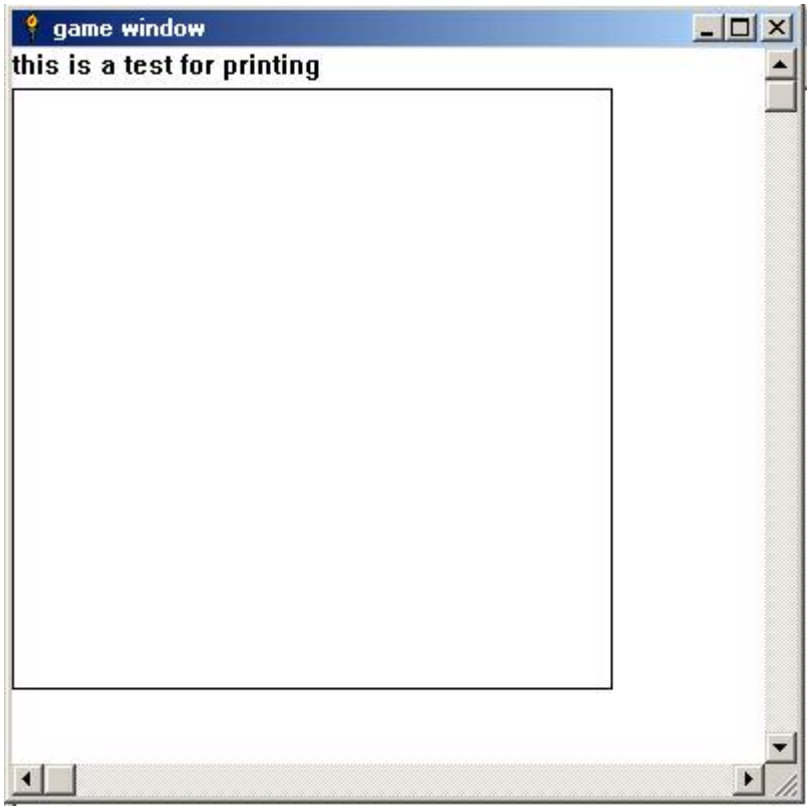
البرنامج اعلاه ليس برنامج للعبة حاسوب و لكن يمكن تحويله الى برنامج لعبة عن طريق التوضيحات الموجودة في هذا الدرس و هناك برنامج في بداية هذا الدرس يشرح كيفية تكوين لعبة حاسوب بسيطة.

كيفية كتابة النقاط (scores) على النافذة:

في بعض الالعاب قد تحتاج الى كتابة النقاط التي احرزها المستخدم خلال اللعبة ,و اذا كنت قد صممت اللعبة على انها ستظهر في الـgraphic box, من الافضل ان تكتب النقاط في الـgraphic window:

```
WindowWidth=400
WindowHeight=400
graphicbox #w.g,0,20,300,300
statictext #w.t,"        ",0,0,200,20
open "game window" for graphics as #w
print #w,"trapclose [quit]"
print #w.t,"this is a test for printing scores"
wait

[quit]
close #w
end
```

و الناتج في المثال السابق سيكون كالتالي:



النافذة في الشكل اعلاه هي الناتج من البرنامج السابق ,لاحظ اننا استعملنا static text للكتابة و كتبنا العبارة (this is a test for printing) في الـgraphic window وذلك لأنه الامر drawsprites في الالعاب يؤدي الى جعل الكتابة غير واضحة في المنطقة التي يستعمل فيها فاذا استعملت الـdrawsprites في الـgraphic box ستظهر الكتابة غير واضحة و لذلك استعمل في الـgraphic window .

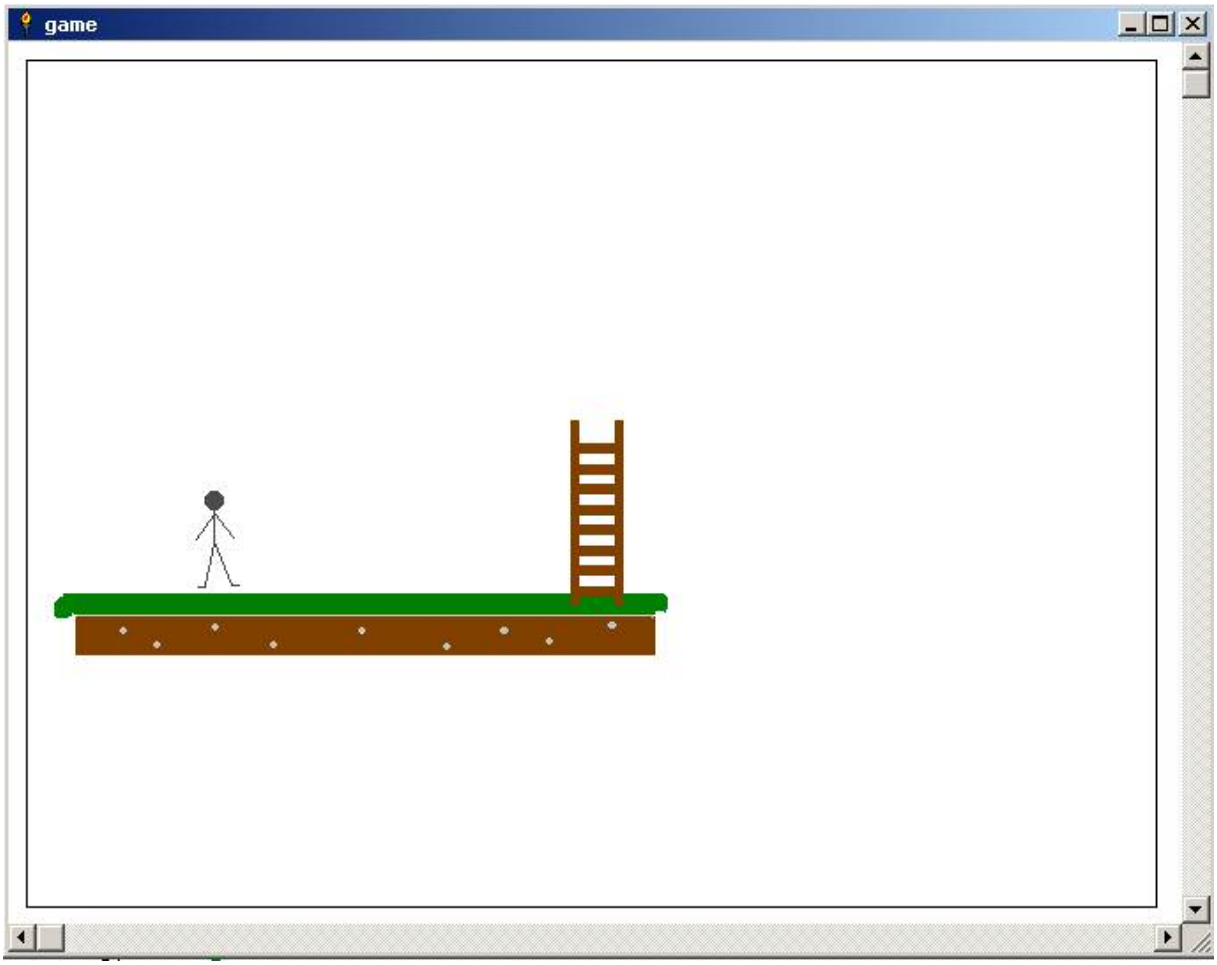
فيزياء الالعاب:
تحتوي الكثير من الالعاب على شئ يسمى فيزياء الالعاب games phisics وتشمل :

١.الجاذبية (gravity).
٢.التصادم (collision).

وقد تم توضيح التصادم (collision) في أحد الدروس السابقة.

يقصد بمفهوم الجاذبية في الفيزياء هي القوة التي يخضع لها جسم معين وتكون متجهة للاسفل,أما مفهوم الجاذبية في الالعاب فلا يختلف كثيرا فقد يكون نحو الاسفل أو قد يكون للجوانب أو الى أي اتجاه كما في بعض الالعاب و لكن في معظم الالعاب تكون الجاذبية متجهة نحو الاسفل وغالبا ما تكون نحو أرضية معينة.
وكيفية برمجة هذا النوع من الفيزياء يكون بالفكرة التالية:
افترض ان هناك لعبة ما تحتوي على شخصية رئيسية و أرضية معينة.فيجب في كل لحظة من اللعبة مراقبة الشخصية الرئيسية للعبة فيما اذا كانت غير مصطدمة مع الارضية,فاذا كانت غير مصطدمة فيجب ان يزداد الاحداثي y بمقدار معين,أما اذا كانت مصطدمة مع الارضية فيجب ابقاء الشخصية الرئيسية للعبة في مكانها على الارضية.

ولا يمكن توضيح هذه الفكرة الا من خلال البرمجة,كالبرنامج التالي الذي يكون الناتج فيه الشكل التالي:



البرنامج التالي سينتج الناتج اعلاه,وهو عبارة عن اختبار للجاذبية الارضية اضافة الى اختبار استعمال التقنيات الاخرى في الالعاب كتسلق السلالم أو الحواجز مثلا,و في البرنامج التالي ستتمكن شخصية اللعبة من تسلق الحاجز بمجرد الضغط على المفتاح w من لوحة المفاتيح عند الوصول الى الحاجز.

```
loadbmp "ladder","f:\lbtech\ladder.bmp"
loadbmp "c1","f:\lbtech\c1.bmp"
loadbmp "floor","f:\lbtech\floor.bmp"
WindowHeight=600:WindowWidth=800
graphicbox #w.g,10,10,640,480
open "game" for graphics as #w
print #w,"trapclose [quit]"
print #w.g,"addsprite gfloor floor"
print #w.g,"addsprite gladder ladder"
print #w.g,"addsprite c c1"


print #w.g,"spritexy gfloor 10 300"
```

```
print #w.g,"spritexy gladder 300 200"
print #w.g,"spritexy c 10 220"

print #w.g,"drawsprites"

timer 100,[timing]
print #w.g,"setfocus"
print #w.g,"when characterInput [keypush]"
wait


[timing]
gosub [chkpos]
gosub [abletojump]


print #w.g,"drawsprites"
wait




[keypush]
h$=Inkey$
if h$="d" then print #w.g,"spritemovexy c 2 0"
if h$="a" then print #w.g,"spritemovexy c -2 0"
if (h$="w" and climb=1) then print #w.g,"spritemovexy c 0 -2"
if (h$="s" and climb=1) then print #w.g,"spritemovexy c 0 2"
if (h$=" " and canjump=1) then jumping=1:jumpcount=20
wait




[chkpos]
print #w.g,"spritecollides c list$"
if instr(list$,"gladder")>0 then
climb=1
else
climb=0
end if
return
```

```
[ableto jump]
scan
if climb=1 then return
if jumping=0 then
print #w.g,"spritexy? c cx cy"
print #w.g,"spritexy c ";cx;" ";cy+5
print #w.g,"spritecollides c ccollides$"
canjump=0
if instr(ccollides$,"gfloor")<>0 then
print #w.g,"spritexy c ";cx;" ";cy
canjump=1
end if
else
print #w.g,"spritexy? c cx cy"
print #w.g,"spritexy c ";cx;" ";cy-5
print #w.g,"spritecollides c ccollides$"
if instr(ccollides$,"gfloor") <>0 then
print #w.g,"spritexy c ";cx;" ";cy
end if
jumpcount=jumpcount-1
if jumpcount<1 then jumping=0
canjump=0
end if
return
```

```
[quit]
close #w
end
```

حيث يتم الاستدلال على انه اذا مازالت الشخصية الرئيسية في الفراغ عند عدم تصادمها مع الارضية و تخصيص قيم مختلفة لبعض المتغيرات.
و في البرنامج السابق الـlabel الذي يسمى [abletojump] هو الذي يتحكم بعملية القفز بواسطة مفتاح المسافة space من لوحة المفاتيح,حيث يختبر في البداية قيمة المتغير climb وهذا المتغير استخدم ليدل على ان الشخصية الرئيسية في اللعبة تتسلق الحاجز فلا تنطبق عليها قوانين الجاذبية الارضية,ثم يختبر المتغير jumping وهو متغير استخدم ليدل على ان الشخصية مازالت في الفراغ ام لا,فاذا كانت قيمة هذا المتغير تساوي 0 كان الجسم في الهواء و لذلك يجب زيادة الاحداثي y بمقدار ما ليهبط الجسم على الارضية,وثم يختبر تصادم الجسم مع الارضية فاذا تصادم سيبقى في مكانه أي على الارضية,أما الاختبار الاخر فهو اذا كانت قيمة jumping تساوي 1 وهي تعني ان الجسم على الارضية لذلك يجب ان يقفز,وبعد ذلك يتم نقصان الاحداثي y ليتم القفز و مع نقصان كل خطوة من الاحداثي y يتناقص متغير الـjumpcount وهو يتحكم بمقدار القفزة و عندما يصل الى القيمة 0 فهذا يدل على ان القفزة انتهت و يجب النزول الى الارضية ولذلك فسيغير قيمة jumping الى 0 ,اضافة الى متغير اخر

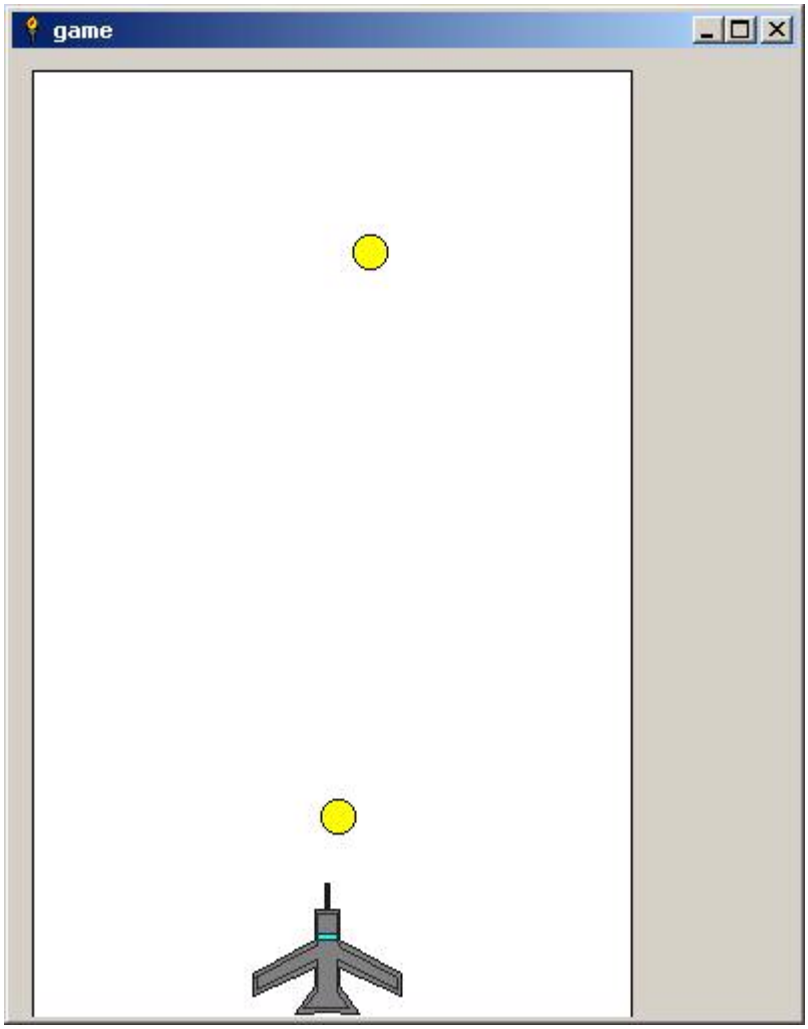
هو canjump وهو متغير يحدد امكانية القفز فحينما يكون الجسم في الفراغ لايجب ان يقفز فتكون قيمته 1 .
أما الـlabel الذي يسمى [chkpos] فهو يستخدم لاختبار امكانية تسلق الحاجز عن طريق المتغير climb .

وهناك نوع اخر من الالعاب مثل العاب الطائرات المقاتلة و التي يجب فهم كيفية برمجة الصواريخ المنطلقة من الطائرة و الخطوة الاولى تأتي عندما يضغط المستخدم على مفتاح اطلاق الصواريخ و ربما يكون مفتاح المسافة space, حيث يتم وضع الـsprite الذي يمثل الصاروخ في المكان الذي يمثل مقدمة الطائرة عن طريق الامر spritexy و الخطوة الثانية هي تحريك الصاروخ عن طريق الامر spritetravelxy, ولكن في هذه الحالة لن تستطيع الطائرة اطلاق عدة صواريخ متتالية لأن عدد الصواريخ هو 1 ويمكن اضافة مجموعة من الـsprites اعتمادا على رقم الـindex أي جعل جميع الـsprites بنفس الاسم و لاتختلف جميعها الا برقم يمثل رقم الصاروخ وهو رقم الـindex, ويزداد كلما ضغط المستخدم المفتاح space .
و البرنامج التالي يوضح ذلك:

```
loadbmp "p1","d:\ plane.bmp"
loadbmp "r","d:\ rocket.bmp"
WindowHeight=510
WindowWidth=400


px=150:py=400:rx=150:ry=470:index=0

graphicbox #w.g,10,10,300,500
open "game" for window as #w
print #w,"trapclose [quit]"

print #w.g,"addsprite p p1"
print #w.g,"spritexy p ";px;" ";py

for i=1 to 10
print #w.g,"addsprite r";str$(i);" r"
print #w.g,"spritexy r";str$(i);" -100 -100"
next i



timer 56,[main]
print #w.g,"when characterInput [press.key]"
print #w.g,"setfocus"
wait

[main]
print #w.g,"drawsprites"

wait
```

```
[press.key]
if Inkey$="a" then px=px-2
if Inkey$="d" then px=px+2
if px<0 then px=0
if px>220 then px=220
if Inkey$=" " then
rx=px+34:ry=py-30
index=index+1
print #w.g,"spritexy r";index;" ";rx;" ";ry
print #w.g,"spritetravelxy r";index;" ";rx;" ";"10 8 [label]"

if index>=10 then index=1
end if
print #w.g,"spritexy p ";px;" ";py

wait

[label]
for i=1 to 10
print #w.g,"spritexy? r";str$(i);" x y"
if y>=10 then print #w.g,"spritexy r";str$(i);" -100 -100"
next i
goto [main]



[quit]
close #w
end
```

استخدمت الحلقة التكرارية for\next لاضافة 10 أشكال sprites وهي تمثل الصواريخ التي تستخدمها الطائرة,ثم يتم وضعها في موقع غير مرئي من الشاشة و هو الاحداثي 100-,100- في المثال أعلاه,وفي الـlabel الذي يسمى [press.key] يتم التحقق من ضغط المفتاح space من لوحة المفاتيح ثم سيتم زيادة المتغير index الذي يتحكم بالرقم بجانب اسم الـsprite لأنه في مثل هذه الحالات يفضل تسمية الـsprites بنفس الاسم و يلي هذا الاسم رقم ما يشير الى رقم الـsprite ويتم توجيه الـsprites باستخدام الامر spritetravelxy و الذي سيتأكد كل مرة يصل فيها أي sprite الى أعلى النافذة من وجود أي sprite في أعلى النافذة في الـlabel الذي يسمى [label] حيث يتم التأكد من وصول كل الـsprites الى أعلى النافذة و عند وصول أي منها يتم نقله الى احداثي غير مرئي من الشاشة وهو في المثال أعلاه 100-,100-
الناتج من البرنامج السابق يكون كالتالي:



وهناك نوع اخر من الالعاب و التي تتحكم بها بواسطة مؤشر الفأرة حيث يمكن ان تختار بعض الـsprites وتحركها,فبعض الالعاب القديمة كانت مثل هذا النوع من الالعاب,حيث يمكن ان تظهر الشخصية الرئيسية للعبة في مكان ما و هناك ادوات معينة على جانب الشاشة يمكن ان تستعملها لحل لغز ما و يجب ان تختار تلك الادوات عن طريق مؤشر الفأرة و كانت هذه الالعاب تعتمد على التفكير وقد يواجه بعض المبرمجين صعوبة في كيفية اختيار عنصر عن طريق الفأرة لأنه لا يوجد أي أمر لكي يرجع لمتغير ما ماهو العنصر الذي يتصادم مع مؤشر الفأرة حاليا,لذلك يجب اضافة sprite و يجب جعله يتحرك مع مؤشر الفأرة دائما و يمكن ان تجعله مخفيا فتستدل على تصادم الفأرة مع أي عنصر عن طريق تصادم الـsprite المخفي مع العنصر ثم توجيه البرنامج الى بعض الـlabels لتتبع الحدث event الناتج من اختيار هذه الادوات .

تغيير شكل مؤشر الفأرة:

يتم تغيير شكل مؤشر الفأرة الى أشكال مدمجة مع لغة Liberty Basic من خلال الامر cursor حيث يليه نوع المؤشر حسب الانواع التالية:

NORMAL = the default pointer
ARROW = the standard Windows arrow
CROSSHAIR = a + shaped pointer
HOURGLASS = the Windows hourglass

TEXT = the text insertion I-beam

حيث يتم كالتالي:

cursor normal

معرفة نوع اخطاء البرنامج عن طريق البرمجة و تجاهل الاخطاء في البرنامج:

كل خطأ في البرنامج يحمل رقم ما,و يحمل قيمة مقطعية لمعرفة ماهو نوع الخطأ و عند حصول أي خطأ سيحتوي المتغيرين $Err و Err نوع الخطأ و رقمه,ويمكن معرفة نوع الخطأ من طبع محتويات هذين المتغيرين و قد يحتوي هذان المتغيران على القيم العددية و المقطعية التالية حسب نوع الخطأ:

3	RETURN without GOSUB
4	Read past end of data
8	Branch label not found
9	Subscript out of range
11	Division by zero
53	OS Error: The system cannot find the file specified.
58	OS Error: Cannot create a file when that file already exists.
55	Error opening file
52	Bad file handle
62	Input past end of file

وقد لايكون هناك رقم للخطأ ففي هذه الحالة سيكون رقم الخطأ 0 .
يمكن توجيه البرنامج الى label معين عند حصول خطأ ما و ذلك عن طريق الجملة التالية:

on error goto [label]

وتوضع هذه الجملة في بداية البرنامج.
حيث الـ[label] هو الـlabel المراد التوجه اليه عند حصول أي خطأ في البرنامج ويمكن وضع الامر resume لمتابعة تنفيذ البرنامج,ويوضع الامر resume في الـlabel الذي يتوجه اليه البرنامج عند حصول خطأ في البرنامج و هذا الـlabel موجود في جملة on error goto [label] .

تغيير عنوان النافذة Main Window:

يستخدم الامر titlebar لتغيير عنوان الـMain Window كل لحظة كما في البرنامج التالي الذي يطبع الوقت الحالي على شريط العنوان titlebar للنافذة Main Window :

```
[main]
    if time$ <> time$() then
       time$ = time$()
       titlebar time$
    end if
    scan
    goto [main]
```

ويستخدم هذا الامر للنافذة Main Window فقط.

التعامل مع المنافذ Ports:

بما ان هذا الكتاب هو ترجمة لملف الـhelp في أغلب مواضيعه فلن نتطرق الى تفاصيل الـports بكافة انواعها لأنه الهدف من هذا الكتاب هو تعلم لغة Liberty Basic فقط دون التطرق الى تفاصيل اخرى,و التعامل مع الـports لا يكون بدون معرفة فيجب توفر المعرفة بكل port و حتى الـserial port و الذي يكون استخدامه سهل جدا مع هذه اللغة.وقد يحصل عطل أو خطأ في الحاسبة نتيجة عدم معرفة كيفية برمجة الـports حيث ان لكل مكان في الـport وظيفة خاصة فبعضها يتحكم بنقل البيانات و بعضها يتحكم بمعالجة الاخطاء في البيانات المنقولة و غيرها من المعلومات التفصيلية التي يمكن دراستها من بعض المصادر من الانترنت و بعض كتب الحاسوب.و المعلومات التالية لم يتم تجربتها على الحاسوب و لكنها ترجمة عن ملف الـhelp و بعض المصادر من الانترنت ,و مؤلف هذا الكتاب لا يتحمل مسؤولية أي خطأ أو ضرر أو عطل في أي حاسبة تجرب فيها هذه المعلومات و هذه المعلومات مجرد ترجمة دون تجربتها في الحاسبة لعدم وجود معلومات كافية عنها و لذلك لم تكتب أي برامج أو أمثلة عن هذا الموضوع ولكن اعطيت فقط طريقة التعامل معها كما هو موجود في الـhelp المرفق مع لغة Liberty basic.

By:gameprogrammer

مع ان المعلومات الموجودة عن المنافذ قليلة جدا في ملف الـhelp المرفق مع البرنامج وقد حاولنا الحصول على بعض المعلومات من اللغات الاخرى كلغة QBASIC ولكن مع هذا تبقى المعلومات عن هذا الموضوع محدودة حاليا و لم تجرب على الحاسبة,ولا ندري ان كانت قد تنجم اخطاء مختلفة عن استخدامها لذلك يجب دراسة منافذ الحاسبة لمعرفة كيفية التعامل معها و نظام Windows لايحتوي على أي صفة تحذر المبرمج عن استعمال أي منفذ قد يستعمله جهاز اخر,فمثلا اذا كان هناك كاميرا رقمية موصلة الى الـ serial port (المنفذ التسلسلي) وكانت حاليا تنقل المعلومات الى الحاسبة و مثلا هناك مبرمج يريد ان يقوم باستخدام نفس المنفذ في برنامجه لنقل المعلومات فسيحصل خطأ لأن المعلومات ترسل بصورة منظمة بين الكاميرا الرقمية و الحاسبة, و نظام Windows لا يحذر المبرمج من استعمال منفذ قد تستعمله أجهزة اخرى حاليا لذلك يجب الحذر عند استعمال المنافذ المختلفة في الحاسبة لأنه قد يحصل عطل أو خطأ,ولم نجرب هذه الاوامر حتى الان و لكن اردنا التنويه عنها هنا و ذلك لتكون كمعلومات اضافية تعود بفائدة على من يعرف كيفية استخدام هذه المنافذ و كل المعلومات التالية هي عبارة عن ترجمة لملف الـhelp المرفق مع لغة Liberty Basic .

التعامل مع المنافذ التسلسلية serial ports:

يوجد منفذين تسلسليين في الحاسبة و هما com1 و com2 و تتميز بأنها بطيئة في نقل البيانات نوعا ما,وهي تنقل المعلومات بمعدل bit في كل فترة نقل.ويستفاد من المنافذ التسلسية في ربط بعض الاجهزة الخارجية للحاسبة مثل الـmodem الخارجي. والخطوة الاولى في استعمال أي serial port هي فتح الـport عن طريق open و تحديد سرعته و عدد الاشارات بالثانية و عدد الـbits المنقولة في كل مرة اضافة الى بعض الاختيارات الاخرى.
حسب الصيغة التالية:

OPEN "COMn:baud,parity,data,stop" for random as #handle

حيث n في COMn هي رقم الـserial port وقد يكون ١ أو ٢ ,و الـbaud هو عدد الاشارات بالثانيةbits per second,و الـparity هو نظام لاختبار البيانات المنقولة,و الـ stop يكون أما ١ stop bit أو ٢ stop bits,وتأتي بعد هذا بعض الاختيارات الاختيارية أي يمكن كتابتها أو اهمالها والـdata تمثل عدد الـbits التي نريد ان يتكون الحرف الواحد منها.

الـn تمثل رقم الـport حيث ١ هو com1 و ٢ هو com2

الـbaud يمكن أن يأخذ القيم التالية:

By:gameprogrammer

٧٥ ١١٠ ١٥٠ ٣٠٠ ٦٠٠ ١٢٠٠ ٢٤٠٠
١٨٠٠ ٢٤٠٠ ٤٨٠٠ ٩٦٠٠ ١٩٢٠٠ ٣٨٤٠٠
٥٧٦٠٠ ١١٥٢٠٠

حيث انه اذا كان هناك جهاز يدعم سرعة غير موجودة في الاعداد اعلاه يمكن اختيار سرعة أكبر لنقل البيانات.

الـparity يأخذ احد الاختيارات التالية:

N	No parity
E	Even parity
O	Odd parity
S	Space parity
M	Mark parity

الـdata يأخذ احدى القيم التالية:

٥ bits long
٦ bits long
٧ bits long
٨ bits long

اذا اخترت الرقم ٨ مثلا فسيعتبر نظام Windows ان كل ٨ bits تشكل حرف واحد.

الـstop يأخذ القيم التالية:

١
٢

حيث ١ هو ١ stop bit و ٢ هي ٢ stop bits . وهو يمثل فترة الانتظار بين نقل كل bit و اخر أي سينقل البرنامج الـbit رقم ١ ثم ينتظر فترة تعادل نقل ٢ bits أي (٢ stop bits) أو ١ bit أي (١ stop bit) لكي ينقل الـbit الاخر,و في اوامر التعامل مع المنفذ التسلسلي لا نتعامل بشكل مباشر مع الـbits بل تطبع جملة ما و البرنامج يقوم بترجمة كل حرف فيها الى عدد من الارقام الثنائية يعادل ما اخترته في قيمة data فاذا كان ٨ ,فسيترجم كل حرف الى ٨ ارقام ثنائية و ينقل كل من هذه الارقام بصورة مستقلة و السرعة المحددة في baud هي عدد الـbits المنقولة بالثانية فاذا كانت ٩٦٠٠ مثلا أي ٩٦٠٠ bit بالثانية,واذا افترضنا ان كل حرف هو ٨ bit فيكون ٨/٩٦٠٠ يساوي ١٢٠٠ حرف بالثانية و اذا افترضت ان كل kByte هو ١٠٢٤ Byte فيكون معدل النقل تقريبا ١ Kbyte بالثانية.

حسب ماموجود من معلومات في ملف الـhelp فالـserial port يتكون من خطوط لنقل البيانات,وهناك خطوط اخرى تتحكم بعملية النقل و يتم التحكم بهذه الخطوط Lines عن طريق اختيارات اختيارية (optional) تضاف الى جملة open و هي:

CSn	Set CTS timeout in milliseconds (default 1000 milliseconds)
DSn	Set DSR timeout in milliseconds (default 1000 milliseconds)
PE	Enable parity checking

RS Disable detection of RTS (request to send)

Other defaults:

DTR detection is disabled
XON/XOFF is disabled
binary mode is the default

حيث الـn في الاختيار الاول تمثل فترة انتظار أحد خطوط النقل وهو Clear To Send ويتمثل فترة الانتظار بوحدة الـ(ملي ثانية milliseconds).و الـn في الصيغة الثانية تتحكم بوقت انتظار الخط data set ready وحسب ماموجود في ملف الـhelp, ان هذه الفترة اذا كانت اكبر من ٠ فهي تسبب توقف البرنامج عن العمل خلال فترة انتظار وصول البيانات لذلك يمكن ان تكون ٠ .أي ان الخط DSR (Data Set Ready) يتحكم بكيفية تصرف البرنامج اثناء فترة انتظار وصول البيانات.
أما لتحديد الحاجز (Buffer) لأكبر عدد يمكن نقله من البيانات فهو عن طريق المتغير Com (لاحظ ان حرف الـC يجب ان يكون بالاحرف الكبيرة) حيث يجب جعل قيمة هذا المتغير تساوي قيمة اكبر عدد ممكن نقله من البيانات و يحدد بوحدة الـByte حيث الـ(kB=1024 Byte) حيث لنقل ١٦ kB من البيانات يجب ان تكون قيمة الحاجز (Buffer) تساوي ١٦*١٠٢٤ وتساوي ١٦٣٨٤ ,ويجب تحديد قيمة هذا المتغير قبل جملة فتح الـ port .

```
Com=16384
open "com2:9600,n,8,1" for random as #commHandle
```

من الافضل ان تجعل قيمة خط الـDSR تساوي ٠ حتى لا يتوقف البرنامج عن العمل عند انتظار وصول البيانات.

```
open "com2:19200,n,8,1,ds0" for random as #commHandle
```

أما ارسال البيانات فيتم عن طريق جملة print و تستخدم مثل طريقة استخدامها في الملفات.حيث لارسال البيانات تستعمل كالتالي:

```
print #handle,data
```

حيث handle هو المقبض الموجود في جملة فتح الـport .
وكما هو موجود في ملف الـhelp فان هناك بعض انواع الـmodem تحتاج الى reset قبل نقل البيانات اليها ولعمل reset لها يمكن ارسال الامر التالي و هو نوع من الاوامر تفهمه بعض انواع من الـmodems :

```
print #handle, "ATZ"
```

لمعرفة عدد الـbytes الموجود حاليا في الـport و المستلمة من الجهاز الموصل الى الـserial port نستخدم الصيغة التالية:

```
variable=lof(#handle)
```

حيث سيخزن الـvariable و الذي يمثل متغير عددي عدد الـbytes المستلمة حاليا و الموجودة في الـ serial port
ولقراءة الـbytes المستلمة عند الـ serial port يمكن استخدام الصيغة التالية:

```
variable$=input$(#handle,n)
```

حيث ستخزن البيانات المأخوذة من الـPort في المتغير $variable ,و n تمثل عدد الـbytes المأخوذة من الـserial port ,حيث يمكن قراءة كل الـbytes الموجود عند الـport عن طريق الصيغة التالية:

```
variable$=input$(#handle,lof(#handle))
```

وبعد انهاء البرنامج أو انهاء التعامل مع الـserial port يجب غلقه باستخدام الجملة close:

close #handle

عند ارسال البيانات من حاسبة ما الى جهاز موصل الى الحاسبة فان العملية تكون ارسال و استقبال للبيانات و عندما ترسل بيانات كثيرة في وقت واحد فهي تتأخر لأنه في كل فترة نقل يجب ان ينقل bit واحد فقط لذلك الـbits الاخرى تنتظر في صفوف, و يمكن معرفة عدد الـbytes المنتظرة في هذه الصفوف (أي الـbytes غير المنقولة حتى الان و هي في الانتظار للنقل) من خلال الجملة txcount .

variable=txcount(#handle)

قد تحصل بعض الاخطاء في الـ serial port ويمكن توجيه البرنامج الى label ما عند حصول خطأ ما,وذلك عن طريق الجملة oncomerror ويأتي بعد هذه الجملة اسم الـlabel المراد التوجه اليه عند حصول خطأ في الـport :

oncomerror [errorlabel]

أما لالغاء التوجه الى أي label عند حصول خطأ فيمكن كتابة الجملة oncomerror بدون أي label:

oncomerror

وعند حصول خطأ هناك ثلاثة متغيرات ستحتوي على قيم تمثل نوع الخطأ و غيره وهذه المتغيرات هي:

ComError$

هذا المتغير يحتوي على وصف الخطأ الحاصل في الـ port .

ComPortNumber

يحتوي هذا المتغير على رقم الـ port الذي حصل فيه الخطأ.

ComErrorNumber

يحتوي هذا المتغير على رقم الخطأ و الذي يختلف بين انواع نظام الـwindows المختلفة.

المنافذ hardware Input/Output ports:

لاتوجد معلومات كافية في ملف الـhelp عن هذه المنافذ,ولكن نعتقد انه لكل جهاز موصل الى الحاسبة port ما يستطيع من خلاله التعامل مع الحاسبة.ويبلغ عدد الـports في الحاسبة ٦٥،٥٣٥ .و مثلا فان منفذ الطابعة المتوازي parallel port هو اسرع من الـserial port و ذلك لأنه ينقل ٨ bit في وقت واحد و ان الموصلات الموجودة في هذا المنفذ ليست جميعها لنقل البيانات و لكن بعضها لنقل البيانات و الاخر يحمل الاشارات التي ترسلها الطابعة (أو أي جهاز يربط على هذا الـport) الى الحاسوب,و لذلك يمكن استخدامه في تحويل الحاسوب الى جهاز يتحكم بالاجهزة المختلفة مثلا عمل ارسال اشارة من الحاسوب الى دائرة الكترونية تتحكم بتشغيل مصباح ضوئي أو تشغيل محرك و غيرها.و لا يتم التعامل مع منفذ الطابعة عن طريق رقم port واحد بل لكل مجموعة من الموصلات في منفذ الطابعة port محدد.و اذا كنت تريد ان تصمم برنامج للتعامل مع أي منفذ من الحاسبة يجب البحث في الانترنت على دليل كامل على الـport الذي ستستعمله لمعرفة كيفية ارسال و استقبال المعلومات منه,لأنه قد يتكون الـport من مجموعة من الـpin (التي تستخدم في الدوائر المتكاملة الـIC) وهذه الـpin تقسم الى انواع فبعضها يرسل بيانات و الاخر يستقبل البيانات من أي جهاز

مربوط بذلك المنفذ و غيرها,لذلك فعند البحث عن هذه المصادر في الانترنت يجب ان تعرف قبل ان تصمم البرنامج أن تتعامل مع هذه المنافذ,لأنه لكل مجموعة من الـpin رقم port خاص فمثلا ان منفذ الطابعة parallel port تقسم الـموصلات فيه الى ٣ مجموعات بعضها تسمى status و الاخرى control و الاخرى output و لكل مجموعة رقم Port محدد تستخدم كلها للتعامل مع المنفذ المتوازي parallel port .

inp(port number)

هذه الجملة تستلم عدد (Byte) ما من الـ port الذي رقمه port number, أي تأخذ البيانات الموجودة في الـ port, و تكون البيانات المأخوذة دائما عبارة عن أعداد وهذه الاشارة عبارة عن المعلومات القادمة من الجهاز الموصل مع الـ Port الذي رقمه port number.

out port number,byte

هذه الجملة تقوم بارسال byte الى الـ port الذي رقمه port number, ويمكن ان يكون هذا الـ byte عبارة عن رقم من ٠-٢٥٥ .

لاتستخدم هذه الجمل مالم تكن متأكدا من معرفة كيفية استعمالها لأنها قد تسبب حدوث اخطاء أو ربما عطل في الحاسبة,أي انه ربما كان النظام Windows يتعامل مع هذه الـ ports ثم يقوم البرنامج الذي صممته مثلا بارسال بيانات الى نفس الـ port الذي يتعامل معه Windows حاليا فهذا سيسبب حصول أخطاء و ذلك لأن نظام Windows لايحتوي على صفة تمكنه من منع البرامج الاخرى من ارسال بيانات الى أي port حتى لو كان النظام Windows يستعمله.

عند استخدام الاوامر inp() و out في البرنامج و تريد توزيع البرنامج الى المستخدمين يجب ارفاق بعض الملفات الموجودة في المجلد الفرعي الموجود في نفس المجلد الرئيسي للغة Liberty Basic, وهذا المجلد يسمى ntport .

ولكن ستختلف الملفات المنقولة الى حاسبات المستخدمين حسب نوع الـWindows :

١. windows 95/98/ME :

ستحتاج الى ملف واحد فقط هو ntport.dll ويجب وضع هذا الملف في المجلد system الموجود في مجلد الـwindows .

٢. windows NT/2000/XP :

لاندري بالضبط ماذا يقصد بالحقوق الادارية (administrative rights) الموجودة في ملف الـhelp عند استعمال windows NT/2000/XP ولكن هناك طريقتين مترجمتين من ملف الـhelp:

١. اذا كان لجميع المستخدمين حقوق ادارية (administrative rights) يمكن ان تستنسخ الملفات ntport.dll و zntport.sys الى المجلد system32 الموجود ضمن مجلد WinNT .

٢. اذا لم يكن لجميع المستخدمين حقوق ادارية (administrative rights) ستحتاج الى تصميم برنامج يقوم بالعمليات التالية:

-استنساخ ntport.dll الى المجلد system32 ضمن المجلد WinNT .

-استنساخ zntport.sys الى المجلد drivers ضمن المجلد system32 الذي يكون ضمن المجلد WinNT.
-اضافة اعدادات جديدة الى الـregistry من خلال الملف ntport2.reg ,ويمكن أن تجد هذا الملف في المجلد ntport الموجود ضمن المجلد الرئيسي للغة Liberty Basic و يكون موجودا مع الملفات الاخرى مثل ntport.dll و zntport.sys ,ونعتقد ان اضافة الاعدادات الى الـregistry تكون عن طريق تشغيل الملف ntport2.reg .
-اعادة تشغيل النظام Windows .

ملاحظات عن استخدام Liberty Basic :

عند اكمال عملية البرمجة و اذا كنت تريد توزيع أو بيع البرنامج الذي قد صممته لا يعقل ان تعطي البرنامج المكتوب الى المستخدمين و اذا حصل هذا فسيحتاج المستخدم الى هذه اللغة لتشغيل هذا البرنامج,ولذلك يوجد الـCompiling أي ترجمة البرنامج الى لغة يفهمها أي حاسوب ولكن الـCompiling يختلف في هذه اللغة لأن البرنامج الناتج ليس ملف واحد فقط لكن مجموعة ملفات,لأنه لا يستخدم Compiling كالذي تشاهده في لغات اخرى أي تحويل البرنامج الى لغة الماكنة,لكنه يستعمل تقنية Virtual Machine حيث يحول ملف لغة الـLiberty Basic الى ملف بالامتداد TKN. حيث يتم تشغيل هذا الملف من خلال برنامج تنفيذي بالامتداد EXE. يجب تغيير اسمه ليكون نفس اسم الملف TKN. حيث يسمى هذا البرنامج بـRun-Time Engine ,وكذلك يجب ارفاق كل الصور و الاصوات المستعملة في البرنامج مع البرنامج المراد توزيعه,وكذلك ارفاق مكتبات الربط الديناميكي DLL و الملفات من نوع SLL. مع البرنامج.
وسيتم توضيح كيفية انجاز هذا بعد شرح بعض القوائم في محرر هذه اللغة.

لترخيص هذه اللغة يجب شراءها عن طريق الانترنت لذلك وبعد شراءها يمكن اختيار enter registration code من قائمة setup ثم ادخال المعلومات التي حصلت عليها عند شرائك لهذه اللغة.كذلك يمكن تصميم الايقونات (icons) عن طريق برنامج icon editor الموجود في قائمة setup .كذلك يمكن التحكم بصفات هذه اللغة من خلال الاختيار preferences من قائمة setup حيث تظهر نافذة تحوي الاختيارات التالية:

القطاع notifications:

الاختيار confirm on exit from Liberty Basic:عند اغلاق محرر هذه اللغة سيجري التأكيد من عملية الخروج من هذه اللغة.
الاختيار display execution complete notice:اظهار رسالة لتنبيه المستخدم عند انتهاء عمل البرنامج و تكون هذه الرسالة في الـMain Window اذا كان للبرنامج هذه النافذة ولم يحذفها المبرمج باستخدام الامر nomainwin.

القطاعstarting up:

الاختيار start Liberty Basic Editor full screen:جعل نافذة محرر هذه اللغة على أبعاد الشاشة.

القطاع load on startup:

الاختيار no file:عدم تحميل أي ملف عند بدء تشغيل محرر هذه اللغة.
الاختيار most recent file:تحميل اخر ملف فتح قبل اغلاق محرر هذه اللغة.

الاختيار this file:يتم كتابة اسم الملف المراد تشغيله في مربع النصوص تحت هذا الاختيار.

القطاع compiling:

الاختيار show compile progress dialog:اظهار progress bar ليوضح عملية الترجمة compiling للبرنامج.
الاختيار enable compiler reporting:في بعض الاحيان ينبه محرر اللغة المبرمج عن وجود متغيرين متقاربين بالتسمية مثلا:
المتغير test و المتغير Test فهذان المتغيران يختلفان عن بعضهما,وكذلك بعض التنبيهات الاخرى ولذلك عند تأشير هذا الاختيار سينبهك محرر هذه اللغة عن وجود اخطاء في ألبرنامج.
الاختيار create *.bak file on run/debug :تكوين ملف بامتداد BAK.* في المسار الموجود في مربع النصوص اسفل هذا الاختيار في وقت التنفيذ run أو وقت تصحيح أخطاء البرنامج debug.و هذه الملفات تكون نسخة من البرنامج الذي ينفذ حاليا و ذلك لتجنب فقدان أو عطل البرنامج فيكون هناك نسخة منه.

By:gameprogrammer

القطاع environment:

الاختيار use syntax coloring:استخدام ألوان معينة لتوضيح عبارات البرنامج فمثلا المتغيرات يكون لها لون ما و مصطلحات هذه اللغة يكون لها لون اخر و هكذا.
الاختيار add kill basic apps to all windows:يتم اضافة هذا الاختيار وذلك لامكانية اغلاق النوافذ عند حصول أي خلل في البرنامج و عدم امكانية اغلاق البرنامج.
الاختيار main window columns/rows:يمكن تعيين الابعاد الافتراضية لنافذة الـMain Window من خلال هذا الاختيار.
الاختيار source filename extension :يمكن كتابة الامتداد الذي سيكون لكل البرامج التي كتبت بهذه اللغة بدلا من ان تكون BAS. وهذا مفيد عندما يكون للمبرمج عدة أنواع من لغة Basic ويريد جعل ملفات هذه اللغة بامتداد اخر غير الامتداد من نوع BAS.

تكوين الملفات التنفيذية من نوع EXE. :

ليس هناك طريقة لترجمة البرنامج الى لغة الماكنة في هذه النسخة من Liberty Basic ولكن يوجد ما يسمى بملفات TKN. و ملف الـRun Time Engine, و ملف الـRun Time Engine موجود في نفس مسار ملفات هذه اللغة و هو من نوع EXE. و يسمى run401.exe (بالنسبة للغة Liberty Basic التي نناقشها هي Liberty Basic 4.01).
افترض انه قد صممت برنامج ما و تريد تحويله الى صيغة يستطيع المستخدم العادي تشغيلها دون الحاجة لهذه اللغة و اسم هذا البرنامج test.bas فلتحويله يجب القيام بالخطوات التالية:

١.تحويل الملف الى ملف من نوع TKN. وذلك من خلال الاختيار make *.TKN file من خلال القائمة run, تظهر نافذة يجب كتابة اسم لملف الـTKN. الجديد.

٢.تكوين folder جديد و تسميته باسم ما,واستنساخ ملف الـTKN. فيه ثم استنساخ ملف الـrun401.exe الموجود في نفس مسار هذه اللغة ووضعه في هذا الـfolder ثم تغيير اسم الملف المستنسخ (run401.exe) الى نفس اسم ملف الـTKN.
٣.اذا كنت قد استخدمت ملفات من أي نوع تابعة للبرنامج و يحتاجها البرنامج اثناء عمله مثل ملفات الاصوات و ملفات الـBMP. يجب استنساخها الى نفس الـfolder الجديد.
٤.استنساخ الملفات التالية (الموجودة في نفس مسار هذه اللغة) الى الـfolder الجديد:

vbas31w.sll
vgui31w.sll
voflr31w.sll
vthk31w.dll
vtk1631w.dll
vtk3231w.dll
vvm31w.dll
vvmt31w.dll

يتم تشغيل البرنامج من قبل أي مستخدم بتشغيل ملف الـRun Time Engine الذي من المفترض ان اسمه قد غيرته الى نفس اسم ملف الـTKN.

يمكن اضافة البرامج المراد تنفيذها مباشرة الى قائمة run في محرر هذه اللغة على أن تكون هذه البرامج من النوع TKN. ,وذلك من خلال اختيار external programs من القائمة setup ثم كتابة اسم البرنامج فسيضاف الى قائمة run ويمكن تشغيله بمجرد اختياره.

وهناك اختيار في قائمة run وهو go to branch label حيث بمجرد اختياره تظهر على يسار الشاشة قائمة كل الـlabels الموجودة في البرنامج و تستطيع الانتقال الى أي منها بمجرد اختياره.

و الاختيار الاخر في نفس هذه القائمة هو lite debug أي اختبار البرنامج للكشف عن أي اخطاء برمجية ,و الاختيار الاخر هو debug حيث تستطيع اختبار البرنامج و الكشف عن قيمة كل متغير.