لا تبدأ فورًا بكتابة الكود، استخدم المكتبات



تعتبر المكتبات من أساسيات لغة البرمجة C، حيث أنَّ اللغة نفسها لا تدعم سوى الميزات والاقترانات الأساسية التي تحتاجها. فمثلًا لغة C لا تحتوي اقترانات الدخل-الخرج اللازمة للقراءة من لوحة المفاتيح والكتابة على الشاشة. وبالتالي أي عملية غير اساسية (كعملية الدخل-الخرج) يجب أن يتم برمجتها من قبل مبرمج. وفي حال كان هذا الاقترانات أو الكود مفيد لبرامج أخرى، يتم عادة وضعه في مكاتب ليكون استخدامه أسهل لاحقًا.

أثناء حديثنا عن لغة البرمجة C، استعرضنا مكتبة واحد فقط حتى الآن وهي مكتبة الدخل-الخرج الأساسية (**stdio**). والتعليمة **include#** في بداية البرنامج هي التي تعطي الأمر للمترجم (**compiler**)بتحميل المكتبة من ملف الترويسة (**header file**)

الذي يكون بعنوان **stdio.h**. حيث أن مصممي لغة البرمجة C يقومون بإضافة مكتبات أساسية مثل مكتبات الدخل-الخرج، مكتبات للاقترانات الرياضية، مكتبات للعمليات الزمنية، ومكتبات للعمليات الشائعة على النصوص. أيضًا يمكنك البحث على الإنترنت أو في أحد المراجع عن الإصدار القياسي للمكتبة **C89** وملاحظة التحديثات والزيادات التي أضيفت في الإصدار **C99**.

تستطيع أنت أيضًا أن تُصمم مكتبة. وعند تصميمك لمكتبة تكون قد قسمت برنامجك إلى وحدات يسهل الوصول إليها، وبالتالي تكون قد سهلت من عملية تضمين نفس الكود البرمجي في برامج أخرى لاحقًا، وأيضًا قللت من عدد الملفات الخاصة بالبرنامج وبالتالي سهلت من عملية قراءته وفحصه وتشغيله.

من أجل تضمين الاقترانات الخاصة بمكتبه ما وتحميلها من ملف الترويسة، يجب إضافة **#include** في بداية السطر البرمجي. بالنسبة للمكاتب القياسية يجب وضع اسم ملف الترويسة الموافق للمكتبة بين علامتي < >. أما بالنسبة للمكاتب الخاصة بك أي التي صممتها أنت، فيمكن وضع الاسم بين علامتي تنصيص " ". وعلى عكس باقي السطور البرمجية، لا داعي لوضع فاصلة منقوطة " ; " بعد تضمين المكتبة. والسطرين الآتيين يوضحان ما تم شرحه:

```
\"include <math.h> #include "mylib.h#)\
```

يجب أن يقدم مصدر البرمجة الشامل التعليمات التي تحتاجها لكتابة مكتبة خاصة بك باستخدام لغة البرمجة C. والتوابع التي سوف يتم كتابتها لن تختلف أبدًا سواءً كانت في المكتبة أو في برنامجك الرئيسي (**main program**).

الاختلاف يكون فقط في أنك سوف تقوم بعملية الترجمة (**compile**) لكل منهما على حدة في شيء يدعى ملف الهدف (**object file**) والذي ينتهي اسمه بـ (**.o**)، وسوف تنشأ ملف آخر يدعى بملف الترويسة ينتهي اسمه بـ (**.h**) والذي يحتوي نماذج التوابع المتوافقة مع كل تابع موجود في المكتبة. وهو أيضًا الذي سوف تشير إليه في تعليمة #include التي ستبدأ فيها كل برنامج يستخدم هذه المكتبة. وسوف تضمّن ملف الهدف (**object file**) في المترجم (**compiler**) في كل مرة تقوم بترجمة البرنامج.

كل الخواص التي استعرضناها إلى الآن تعتبر خواص قياسية موجودة في لغات البرمجة الأخرى، والآن سوف نتطرق إلى كيف تتعامل لغة البرمجة C مع ذاكرة حاسوبك.

بعض المعلومات عن المؤشرات في لغة C:

عندما يتم تحميل برنامجك على ذاكرة حاسوبك (وهنا نقصد تحديدًا ذاكرة الوصول العشوائي **RAM**)، كل جزء في البرنامج يكون مرتبطًا بعنوان في الذاكرة. وهذا يشمل المتغيرات التي يستخدمها لتخزين قيم معيَّنة (**Variables**) . وكل مرة يستدعي برنامجك اقتران ما، فهو يقوم بتحميل الاقتران وكل البيانات المرتبطة به في الذاكرة وذلك لفترة كافية لتشغيل الاقتران والحصول على نتيجة منه. وفي حال قمت بتمرير بيانات إلى الاقتران، تقوم لغة البرمجة C بأخذ نسخة من البيانات التي قمت بتمريرها لاستخدامها في الاقتران.

أحيانًا عندما تقوم بتشغيل اقتران ما، وفي حال أردت القيام بتغيير دائم للبيانات في موقعها الأصلي في الذاكرة، وبما أن لغة C تقوم بعمل نسخ للبيانات (للبارامترات) لاستخدامها في الاقتران، سوف تبقى البيانات الأساسية (التي أردت أن تجري تغييرات عليها) ثابتة لأن التغييرات التي أجريتها قد تم إجراؤها على النسخة وليس على البيانات الأصلية. لذلك في حال أردت أن تجري تغييرًا دائمًا، عليك أن تمرر مؤشر عن العنوان في الذاكرة للبيانات (**pointer**) أي أن تقوم بتمرير مكان المعلومة (التمرير بالمرجع) وليس تمرير القيمة نفسها (التمرير بالقيمة).

تُستخدم المؤشرات بشكلٍ واسعٍ جدًا في لغة C. لذلك في حال أردت أن تستخدم لغة البرمجة C بشكل متقن، عليك أن تمتلك دراية جيدة عن كيفية استخدام المؤشرات. والمؤشر هو عبارة عن متغير كباقي المتغيرات، الغرض منه هو تخزين عنوان الذاكرة لبيانات أخرى. وللمؤشر أيضًا نوع بيانات، وبالتالي فهو قادر على تمييز الوحدات الثنائية **bits** التي تعبر عن موقع في الذاكرة.

عندما تنظر إلى متغيرين اثنين بجانب بعضهما في لغة C، فإنك لن تستطيع أن تتعرف أيّ منهما هو المؤشر، حتى بالنسبة للمبرمجين المحترفين فهذا لن يكون سهلًا. ولكن عندما تقوم أنت بإنشاء مؤشر ما، في هذه الحالة سوف يكون واضحاً لك، لأنه يجب أن تسبقهُ مباشرة بالعلامة (*) قبل اسم المؤشر فورًا. وهذا يعرف بالعامل الغير مباشر في لغة C. الكود التالي سوف يقوم بإنشاء متغير(**i (variable**، ومؤشر (**p (pointer** :

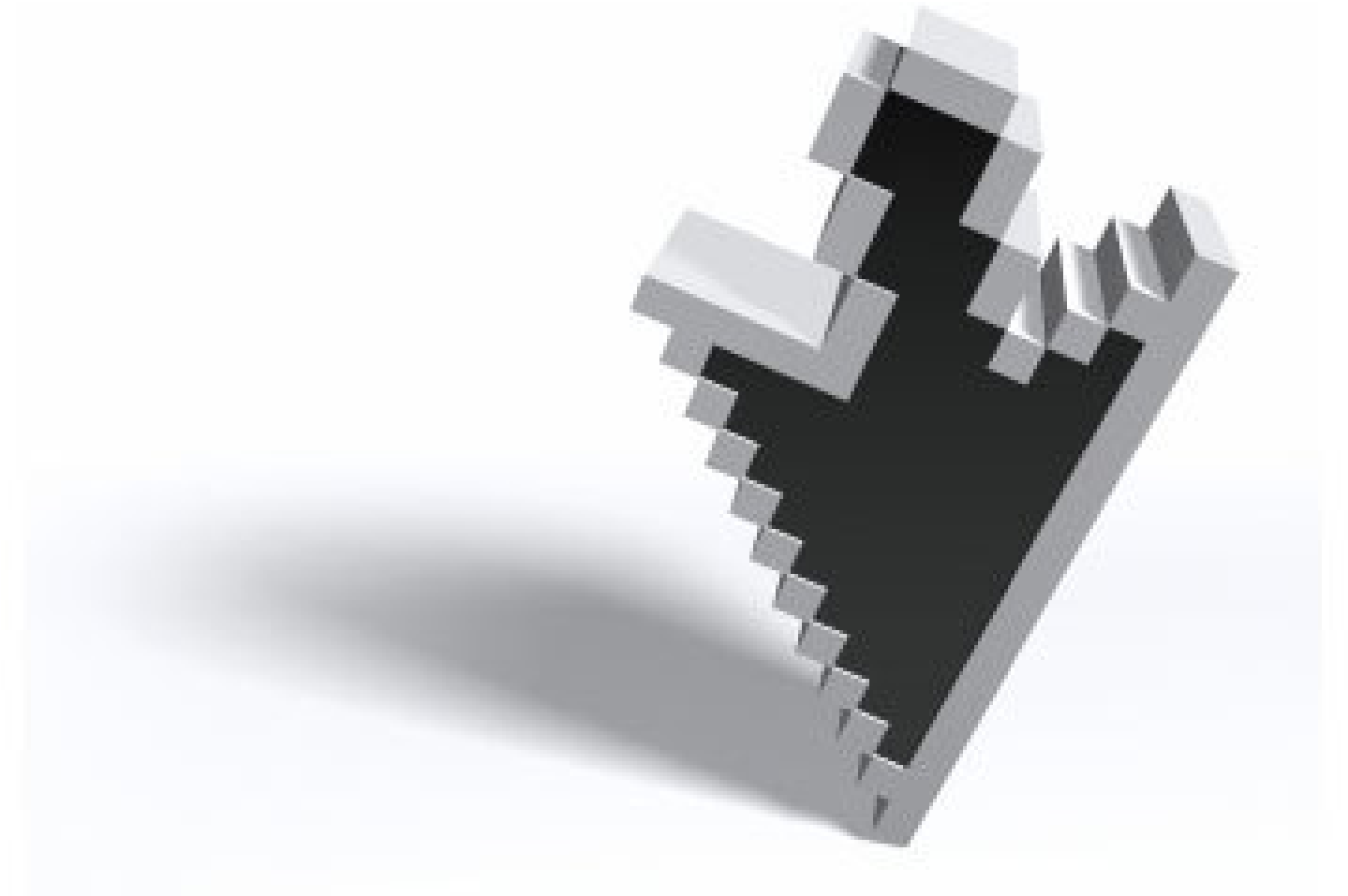
(\;int i; \\int *p)\

لا توجد قيم لأيّ من المتغيرين حاليًا، سوف نقوم بإسناد قيمة للمتغير i، ثم نعين p كمؤشر لموقع الذاكرة لـ i (أي أن p سوف يقوم بتخزين عنوان موقع الذاكرة الخاصة بالمتغير i) :

(\;i = 3; p = &i)\

يمكنك ملاحظة العلامة (&) المستخدمة كعامل عنونة والموضوعة مباشرةً قبل i للدلالة على عنوان الذاكرة لـ i. ليس عليك أن تعرف ما هو هذا العنوان لكي تقوم بإسناده إلى p، وهذا جيد لأنَّه على الأرجح سوف يتغيَّر في كل مرة تقوم بتشغيل البرنامج، عوضًا عن ذلك، عامل العنونة سوف يتولى مهمة تحديد عنوان الذاكرة الخاص بالمتغير i عندما يكون البرنامج في حالة تشغيل. وفي حال لم تستخدم عامل العنونة (&)،سوف يقوم الإسناد التالي p=i بإسناد عنوان الذاكرة الخاص بالرقم 3 حرفيًا، وليس عنوان الذاكرة الخاص بالمتغير i.

الآن، دعنا نرى كيف يمكنك استخدام المؤشرات في لغة C، وما هي التحديات التي يجب عليك الاستعداد لها.
استخدام المؤشرات بطريقة صحيحة في لغة C:



إذا أردت أن تصبح مبرمجًا محترفًا في لغة C، يجب عليك أن تمتلك خبرة جيدة باستخدام المؤشرات في برنامجك الخاص

عند إنشاء مؤشر، فإنه بإمكانك استخدامه كمتغيِّر من نفس نوع البيانات في العمليات أو عند استدعاء الاقترانات. في المثال التالي، المؤشر لـ i سوف يستخدم بدلًا من المتغير i خلال عملية أوسع. العلامة المستخدمة مع p*) p) تدل على أن العملية يجب أن تستخدم القيمة التي يؤشر عليها المؤشر p في عنوان الذاكرة (وليس العنوان نفسه).

```
(\ ;int b;\\ b = *p + 2)\
```

عملية تقسيم المهام إلى اقترانات يعتبر مستحيل بدون استخدام المؤشرات في لغة C. ولتوضيح ذلك، تخيل أنك انشأت متغير اسمه h والذي يقوم بتخزين طول المستخدم بالسنتيمتر. وقمت أيضًا باستدعاء اقتران (برنامج فرعي) قمت أنت بتصميمه واسميته **setHeight** والذي يطلب من المستخدم أن يقوم بإدخال طوله. وعليه يمكن للكود أن يكون على الشكل التالي:

```
(\;(int h;\\setHeight(h)\
/*يوجد مكلة محتملة هنا */
```

عند استدعاء الاقتران بهذه الطريقة سوف يقوم بتمرير قيمة h والتي تعني هنا الطول الذي قام المستخدم بإدخاله، ولكن عند انتهاء الاقتران، لن يحدث أي تغير في قيمة h لأن الاقتران قد استخدم نسخة عن قيمة h، وقام بحذفها (نظام التشغيل مع المترجم) عند انهاء عمله.

إذا كنت ترغب في تغيير قيمة h نفسها، عليك أولًا التأكد من أنَّ الاقتران يسمح لك بتمرير مؤشر له عوضًا عن تمرير نسخة عن قيمة h.

(وهذا يتم تحديده في السطر الأوّل من الكود البرمجي الخاص بالتابع **setHeight**)، ثم أنَّه سيستخدم المؤشر بدلًا من القيمة المنسوخة كبيانات مرسلة له (لاحظ العامل الغير مباشر):

```
\(setHeight(&h)\;)
/*تمرير العنوان الذاكري للمتحول كبارمتر للتابع */
\(int *p; p = &h; setHeight(p)\ ;)
/*تمرير مؤشر منفصل يدل علىعنوان ذاكرة لمتغير آخر */
```

والطريقة المتبعة في الحالة الثانية (السطر الأخير) تكشف عن تحدٍ شائع عند استخدام المؤشرات. التحدي يتمثل في امتلاك عدة مؤشرات لنفس القيمة. وهذا يعني أنه في حال حدوث أي تغيّر في إحدى القيم سوف تتأثر جميع مؤشراتها في الوقت ذاته. هذا من الممكن أن يكون شيء جيد أو شيء سيئ على حسب المهمة التي تريد تنفيذها في برنامجك. مرة أخرى نذكّر أنَّ التمرس في استخدام المؤشرات يعتبر الأساس في احتراف البرمجة بلغة C. حاول أن تتدرب على استخدام المؤشرات بشكل كبير لكي تصبح قادرًا على مواجهة هذه التحديات البرمجية.

الخواص التي استعرضناها إلى الآن تعتبر خواص قياسية موجودة في لغات البرمجة الأخرى، الآن سوف نلقي نظرة على المطالب الأساسية للغة C من أجل إدارة فعالة للذاكرة.

أهمية إدارة الذاكرة في لغة C :

إن إحدى الميزات التي تجعل من لغة C لغة مرنة هي أنَّ للمبرمج القدرة على خفض مساحة الذاكرة المطلوبة لتشغيل البرنامج. عندما صمّمت لغة C لأول مرة، كانت هذه الميزة مهمَّة جدًا لأنَّ الحاسب آنذاك لم يكن بالقوة التي هو عليها اليوم، وبسبب الطلب المستمر على الإلكترونيات صغيرة الحجم، من الهواتف النقالة إلى الأجهزة الطبية الصغيرة، هناك اهتمام متجدد في إبقاء متطلبات الذاكرة صغيرة لبعض البرمجيات، ولغة C هي اللغة المناسبة لمن يحتاج إلى تحكم كامل في استخدام الذاكرة.

ولفهم أفضل لأهمية إدارة الذاكرة، لننظر في كيفية استخدام البرنامج للذاكرة. عندما تقوم بتشغيل البرنامج، فإنه يتم تحميله مباشرةً إلى ذاكرة برنامجك ويبدأ في العمل عن طريق إرسال واستلام التعليمات من معالج الحاسوب. وعندما يحتاج البرنامج لتشغيل اقتران معين، فإنه يقوم بتحميل هذا الاقتران إلى جزء آخر من الذاكرة طوال فترة عمله، ويقوم بترك هذا الجزء عند انتهاء عمله. كل جزء من البيانات المستخدمة في البرنامج الأساسي تشغل حيز من الذاكرة طوال الفترة التي يكون فيها البرنامج في حالة عمل.

في حال أردت أن تتحكم بشكل أكبر من ذلك، فإنك تحتاج إلى ما يعرف بتوزيع الذاكرة الديناميكي والذي تدعمه لغة C. ,ويمكن تعريف توزيع الذاكرة الديناميكي بالقدرة على حجز حيز معين من الذاكرة وتحرير هذا الحيز في اللحظة التي تنتهي فيها من استخدامه. تمتلك العديد من لغات البرمجة توزيع ذاكرة أتوماتيكي بالإضافة إلى عملية تدعى تجميع القمامة – افراغ مواقع الذاكرة من البيانات التي لم تعد هناك حاجة اليها (**garbage collection**) والذي تقوم بتولي مهام إدارة الذاكرة كجزء من مظام التشغيل. ومع ذلك تسمح لك لغة C (وفي بعض الأحيان تجبرك) بأن تكون واضح أو صريح في عملية حجز الذاكرة عبر هذه المهام الأساسية المدرجة في مكتبة C القياسية:

- **malloc** ـ ـ وهو اختصار لحجز الذاكرة، حيث يستعمل **malloc** لحجز حيز معين مصرح عنه من الذاكرة ومن نفس نوع البيانات التي يحتاج البرنامج لمعالجتها. وعند استخدامك لـ **malloc** تكون قد أنشأت مؤشر يحدد المكان الذي تم حجزه في الذاكرة (**allocated memory**) . وهذا لا يعتبر ضروري لجزء معين من البيانات، مثل متغير وحيد من نوع integer، والذي يتم حجز مكان له في الذاكرة

فور التصريح عنه (كما في **int i**) . على أي حال، إن هذا ضروري جدًا من أجل إنشاء وإدارة هياكل بيانات كالمصفوفات مثلًا. وخيارات الحجز البديلة في لغة C هي **calloc**، والتي تقوم بمسح الذاكرة عندما تكون محجوزة. و أيضًا **relloc**، والتي تقوم بتغيير حجم الذاكرة المحجوزة مؤخرًا.

- **free** – يتم استخدامه للمسح القصري للذاكرة التي تم إسنادها إلى مؤشر سابق.

من أجل استخدام أمثل لـ **malloc** و **free** يجب أن يتم مسح كل ما يتم تخصيصه عند الانتهاء منه. لأنه عند القيام بتخصيص أي شيء حتى في الاقترانات المؤقتة، تبقى الذاكرة ممتلئة حتى يقوم نظام التشغيل بتحرير هذه المساحة. ومن أجل التأكد من أن المساحة خالية وجاهزة للاستعمال الفوري، يجب أن يتم الغاء حجزها قبل أن يتم الخروج من الاقتران الحالي. وهذه الإدارة للذَّاكرة تعني أنَّه يمكنك تقليص أثر هذا الاقتران إلى الحد الأدنى وما يدعى بتسريب الذاكرة. وتسريب الذاكرة هو خلل في البرنامج يقوم باستهلاك مساحة بشكل مستمر إلى حين عدم بقاء أي مساحة ممكنة للاستخدام، مما يؤدي إلى تعليق البرنامج أو انهياره (بسبب نفاذ المساحات الخالية من الذاكرة). ومن جهة أخرى، يجب عليك أن لا تتسرع في عملية مسح الذاكرة وبالتالي تفقد معلومات قد تحتاجها لاحقًا في نقس الاقتران.

من خلال هذه السلسلة، لقد تعلمت بعضًا من البنى والمفاهيم الأساسية عن البرمجة باستخدام لغة C. من خلال الاطلاع أولًا على تاريخ هذه اللغة، خواصها والجوانب التي تتشارك فيها مع لغات البرمجة الأخرى، والخواص الأساسية التي تجعل منها لغة فريدة وخيارًا جيدًا لاستخدامها كلغة برمجة.

- التاريخ: 2016-08-14
- التصنيف: تكنولوجيا

#البرمجة #لغات البرمجة #البرمجة بلغة c

المصادر

- how stuff works
- الصورة

المساهمون

- ترجمة
  - محمد اسماعيل باشا
- مُراجعة
  - أمجد هواش
- تحرير
  - طارق نصر
  - أنس الهود
- تصميم
  - علي كاظم
- نشر

- سارة الراوي