

السلام عليكم ورحمة الله وبركاته

Week 2 - (Arrays)

محتوى المحاضرة:

- 1. Compiling
- 2. Debugging
- 3. help50 and printf
- 4. debug50
- 5. check50 and style50
- 6. Data Types
- 7. Memory
- 8. Arrays
- 9. Strings
- 10. Command line arguments
- 11. Readability
- 12. Encryption

Compiling -1

كما تعلمنا في المحاضرة السابقة syntax الخاصة بلغة C لا يفهمها الكمبيوتر ولكن يجب استخدام Compiling لتحويلها إلى binary system حتى يفهمها الكمبيوتر.
عن طريق لغة C نستخدم الأمر التالي لعمل compiling

```
clang hello.c  
./a.out
```

الأمر clang يقوم بعمل Compiling والملف a.out ما هو إلا إختصار ل assembler output وكما شرحنا

سابقا يمكننا استخدام الامر make لتسهيل كتابة الأمر وتغيير أسم ملف output .
إذا أردنا استخدام الأمر clang بدلا من make عند استدعاء مكتبة CS50's library يجب استخدام الأمر كالتالي

```
clang hello.c -lcs50  
  
./a.out
```

حيث أن -l تعني link in ملف cs50

عند استخدام الأمر make أو clang يقوم الكمبيوتر بتحويل source code إلى machine code عن طريق عدة خطوات كالتالي

```
preprocessing  
  
compiling  
  
assembling  
  
linking
```

• Preprocessing

في هذه الخطوة يقوم بالنظر إلى الكود الموجود ببدايته # ويقوم بإستبداله بالملف الأصلي له كالتالي

```
#include <cs50.h>  
#include <stdio.h>  
  
int main(void)  
{  
    string name = get_string("What's your name?\n");  
    printf("hello, %s\n", name);  
}
```

```

...
string get_string(string prompt);
int printf(string format, ...);
...

int main(void)
{
    string name = get_string("What's your name?\n");
    printf("hello, %s\n", name);
}

```

• Compiling

يقوم بتحويل source code إلى Assembly code وهي أقرب الأكواد إلى أوامر Binary التي يمكن أن يفهمها CPU وهو عقل الكمبيوتر كالتالي

```

...
main:                                # @main
    .cfi_startproc
# BB#0:
    pushq   %rbp
.Ltmp0:
    .cfi_def_cfa_offset 16
.Ltmp1:
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
.Ltmp2:
    .cfi_def_cfa_register %rbp
    subq    $16, %rsp
    xorl    %eax, %eax
    movl    %eax, %edi
    movabsq $.L.str, %rsi
    movb    $0, %al
    callq   get_string
    movabsq $.L.str.1, %rdi
    movq    %rax, -8(%rbp)
    movq    -8(%rbp), %rsi
    movb    $0, %al
    callq   printf
    ...

```

• Assembling

وهنا يقوم بتحويل الكود السابق إلى machine code وهنا يمكن للcpu تنفيذ الأوامر من خلالها مباشرة وتكون كالشكل التالي

[illegible]

linking •

وهنا يتم دمج جميع files الموجودة معا ال header files مع ال source code كالتالي

```
hello.c      cs50.c      printf.c
```

[illegible][illegible]

Debugging -2

bugs هي الأخطاء الموجودة بالبرنامج و الغير مقصودة و debugging هي الطريقة لمعرفة هذه الأخطاء وحلها.

جاء مصطلح bug من grace hopper من أشهر علماء الكمبيوتر والذي وجد بداخل الماكينة bug والتي أحدثت بعض المشاكل ومن هنا جاء هذا المسمى



help50 and printf -3

ماذا لو لدينا مشكلة ما وظهر لدينا خطأ ولم نفهم ماهو الخطأ هنا جاء فائدة أداة help50 وهي أداة مدعومة من cs50 ويمكن إستخدامها كالمثال التالي buggy0.c يمكنك الدخول على الأمثلة من هنا [examples](#)

```
buggy0.c
1 int main(void)
2 {
3     printf("hello, world\n");
4 }
5

Terminal
$ make buggy0
```

عند الضغط على enter هنا ستظهر المشكلة التالية

```
1 // Buggy example for printf and debug50
2
3 #include <stdio.h>
4
5 int main(void)
6 {
7     printf("hello, world\n");
8 }
9
```

make buggy0
clang -fsanitize=signed-integer-overflow -fsanitize=undefined -ggdb -O0 -std=c11 -Wall -Werror -Wextra -Wno-sign-compare -lm -o buggy0 buggy0.c
make: *** [buggy0] Error 1
buggy0.c:15:15: error: implicitly declaring library function 'printf' with type 'int (const char *, ...)' [-Werror,-Wimplicit-function-declaration]
printf("hello, world\n");
^
buggy0.c:15:15: note: include the header <stdio.h> or explicitly provide a declaration for 'printf'
1 error generated.
make: *** [buggy0] Error 1

يمكننا استخدام الأداة كالتالي

```
$ help50 make buggy0
```

ستكون هذه هي النتيجة

```
$ make buggy0
clang -fsanitize=signed-integer-overflow -fsanitize=undefined -ggdb -O0 -std=c11 -Wall -Werror -Wextra -Wno-sign-compare -lm -o buggy0 buggy0.c
make: *** [buggy0] Error 1
buggy0.c:15:15: error: implicitly declaring library function 'printf' with type 'int (const char *, ...)' [-Werror,-Wimplicit-function-declaration]
printf("hello, world\n");
^
buggy0.c:15:15: note: include the header <stdio.h> or explicitly provide a declaration for 'printf'
1 error generated.
make: *** [buggy0] Error 1
$ help50 make buggy0
clang -fsanitize=signed-integer-overflow -fsanitize=undefined -ggdb -O0 -std=c11 -Wall -Werror -Wextra -Wno-sign-compare -lm -o buggy0 buggy0.c
make: *** [buggy0] Error 1
buggy0.c:15:15: error: implicitly declaring library function 'printf' with type 'int (const char *, ...)' [-Werror,-Wimplicit-function-declaration]
printf("hello, world\n");
^
buggy0.c:15:15: note: include the header <stdio.h> or explicitly provide a declaration for 'printf'
1 error generated.
make: *** [buggy0] Error 1
Asking for help...
buggy0.c:15:15: error: implicitly declaring library function 'printf' with type 'int (const char *, ...)' [-Werror,-Wimplicit-function-declaration]
printf("hello, world\n");
^
Did you forget to #include <stdio.h> (in which printf is declared) into your file?
```

هنا يتم شرح المشكلة وتوضيحها باللون الأصفر ويقترح الحل لها وهكذا ...

مثال آخر:

أنظر المثال buggy2

```
buggy2.c
1 // Buggy example for printf and debug50
2
3 #include <stdio.h>
4
5 int main(void)
6 {
7     for (int i = 0; i <= 10; i++)
8     {
9         printf("#\n");
10    }
11 }
```

هنا عند كتابة الكود make لا يظهر Error ولكن عند تنفيذ البرنامج نجد أن output غير المطلوب نحن نريد طباعة هذا الرمز 10 مرات ولكن يتم طباعته 11 مرة وهنا لدينا خطأ ولكنه logical error في صياغة البرنامج وليس مجرد خطأ syntax
يمكننا استخدام الأمر printf لمعرفة الخطأ كالتالي

```
buggy2.c x +
1 // Buggy example for printf and debug50
2
3 #include <stdio.h>
4
5 int main(void)
6 {
7     for (int i = 0; i <= 10; i++)
8     {
9         printf("i is now %i: ", i);
10        printf("#\n");
11    }
12 }
13
Terminal x +
$ make buggy2
clang -fsanitize=signed-integer-overflow -fsanitize=undefined -g -g
-Mahadaw buggy2.c -lcrypt -lcs50 -ln -o buggy2
$ ./buggy2
i is now 0: #
i is now 1: #
i is now 2: #
i is now 3: #
i is now 4: #
i is now 5: #
i is now 6: #
i is now 7: #
i is now 8: #
i is now 9: #
i is now 10: #
$
```

وهنا يكون الأمر printf مساعد لنا لرؤية الخطوات بشكل أدق وهنا سنجد أن الأمر يتم تشغيله 11 مرة ولحل هذه المشكلة نستبدل $i \Rightarrow 10$ ب $i > 10$ فقط ولكننا يمكننا إعتبار أن طريقة printf هي طريقة قديمة لحل المشكلة ومعرفتها ويمكن إستبدالها بالأداة التالية

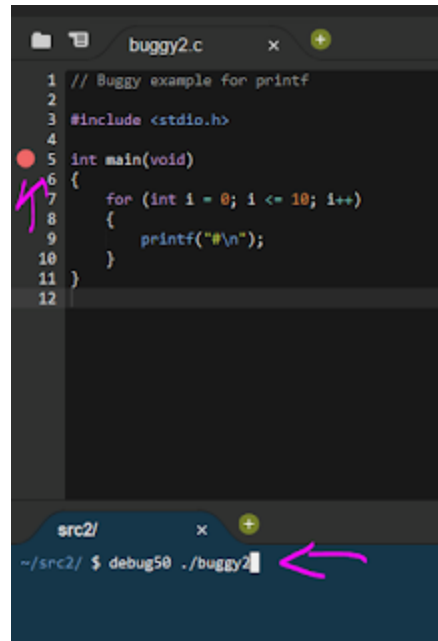
debug50 -4

هنا نستخدم الأداة debug50 وهي مدعومة في CS50 IDE وليس CS50 sandbox حيث أن CS50 IDE يزيد عن CS50 Sandbox في بعض الأدوات والمميزات مثل debugging .

يمكنك الدخول من هنا على [CS50 IDE](#)
أنظر الفيديو لشرح كيف يتم فتحها على cs50 ide

دعونا نجرب المثال السابق ولكن بإستخدام debug50

في البداية سنجد أن الملف بداخل folder يسمى src2 لذا سنستخدم الأمر cd كما شرحنا في فيديو shorts week 1 كالتالي



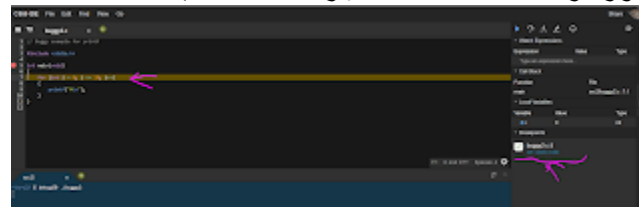
```
1 // Buggy example for printf
2
3 #include <stdio.h>
4
5 int main(void)
6 {
7     for (int i = 0; i <= 10; i++)
8     {
9         printf("#\n");
10    }
11 }
12
```

src2/ x +

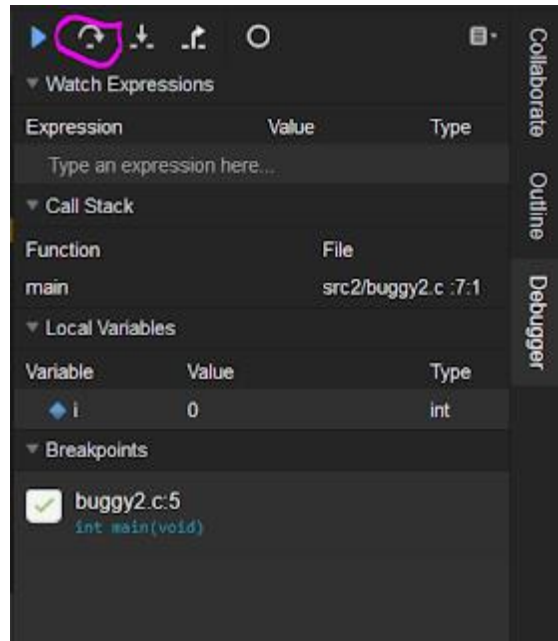
~/src2/ \$ debug50 ./buggy2

قم بالضغط على line وليكن الخامس في هذا المثال لتضع break point لتخبر debugger بالتوقف هنا والتنقل خطوة بخطوة

أكتب الكود كما موضح بالصورة وأضغط enter ستظهر الشاشة التالية



سيقوم بفتح الجزء الموجود على اليسار وهو debugger



ستجد أن المتغير `i` موجود أسفل قسم المتغيرات وقيمته صفر. ستجد أن `break point` أوقفت البرنامج لبدء التشغيل من هذه النقطة وستجد أكثر من رمز في `debugger` المثلث أو علامة التشغيل ستقوم بتشغيل البرنامج والتوقف عند نقطة التوقف التالية أو حتى إنتهاء البرنامج والرمز الآخر الموجود في الصورة السابقة يقوم بتشغيل البرنامج بخطوة بخطوة فيمكننا رؤية ما يحدث بداخل الذاكرة وستجد أنه مع كل خطوة يزيد قيمة `i` وهنا يمكننا إكتشاف الخطأ بدون اللجوء لـ `printf`

ملحوظة : لإغلاق `debugger` نضغط `control+c`

5-5 check50 and style50

من خلال الأمر التالي

`check50 cs50/problems/hello`

يمكننا فحص الكود الخاص بـ `pset 1` كالتالي

```
hello.c
1 #include <stdio.h>
2
3 int main(void)
4 {
5     printf("hello, world\n");
6 }

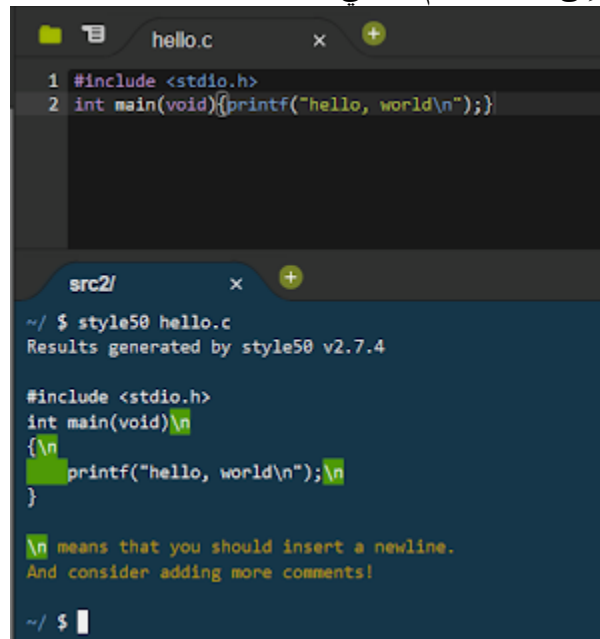
$ check50 cs50/problems/hello
Connecting.....
Authenticating.....
GitHub username: rasha-saber
GitHub password: *****
Verifying.....
Preparing.....
Uploading.....
Waiting for results.....
Results for cs50/problems/hello generated by check50 v3.8.9
1) hello.c exists
2) hello.c compile
3) responds to name tests
   expected "hello", got ""
4) responds to name description
   expected "hello!\n", got ""
To see the results in your browser go to https://submit.cs50.io/check50/0f5eae833d3cc8c9e7267bd97a2815413ee40c2b
~/ $
```

وهنا يخبرك على الخطأ الموجود فيه وهذا خاص فقط بـ `cs50 tests` ولكن في الحياة العملية يمكنك إنشاء `test` خاص بك للتأكد من توفر

المتطلبات التي تريدها في المشروع الخاص بك وفي بعض المشاريع الكبيرة أو بعض الشركات يوجد خطوة من الخطوات ليس فقط عليك إنشاء الكود ولكن عليك إنشاء test للمشروع يكون مرجع لك وللمن يعمل بعدك على هذا المشروع للتأكد من توافر المتطلبات الأساسية له.

style50

هو برنامج يقوم بفحص الشكل العام و المسافات بناء على standard guide لبناء الكود بشكل صحيح ولا يقوم بفحص الأخطاء هو فقط ينظر إلى الشكل العام كالتالي



```
hello.c
1 #include <stdio.h>
2 int main(void){printf("hello, world\n");}

src2/
~/ $ style50 hello.c
Results generated by style50 v2.7.4

#include <stdio.h>
int main(void)\n
{\n
    printf("hello, world\n");\n
}

\n means that you should insert a newline.
And consider adding more comments!

~/ $
```

الملخص

help50	correctness
printf	correctness
debug50	correctness
check50	correctness
style50	style

بعد كل المساعدات السابقة لمعرفة الأخطاء الموجودة فالكود هناك بعض الأخطاء لا يمكنك إكتشافها وهنا الحل
أخذ بعض الراحة
أو ينصحك دكتور ديفيد بإستخدام rubber duck toy والتكلم معها لشرح مشروعك لها بالخطوات .

ليس فقط المطلوب إنشاء مشروع يعمل بشكل صحيح و لكن يجب أن يكون جيد في التصميم لتوفير المال
ومساحة الذاكرة وإستخدام Cpu وهنا يجب أن تكون كفاءة البرنامج عالية وهذا ما سوف نتعلمه في هذا الكورس
وأول خطوة لتعلم كتابة برنامج ذو كفاءة عالية يكون من خلال دراسة Array حيث أن array تساعدنا لحل
المشكلات.

Data Types -6

كما شرحنا سابقا هناك العديد من Data types in c كالتالي

bool	1 byte
char	1 byte
int	4 bytes
float	4 bytes
long	8 bytes
double	8 bytes
string	? bytes
...	

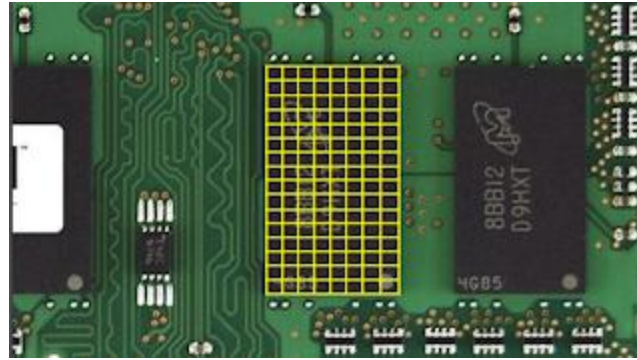
لذلك من خلال كتابة برنامج في C مع إستخدام أى نوع من data types السابقة نحجز جزء من الذاكرة بقيم كل
data type

Memory - 7

بداخل كل كمبيوتر أو هاتف يوجد رقائق تسمى RAM (random access memory) وهي جزء
hardware
تحفظ المعلومات للوقت القصير حتى يتم غلق الكهرباء عن الجهاز
يمكننا حفظ المعلومات للمدى البعيد عن طريق حفظها في Hard drive أو أي وحدة تخزين دائمة ولكن عند فتح
البرنامج Ram تأخذ نسخة منه للمدى القصير وذلك لزيادة السرعة

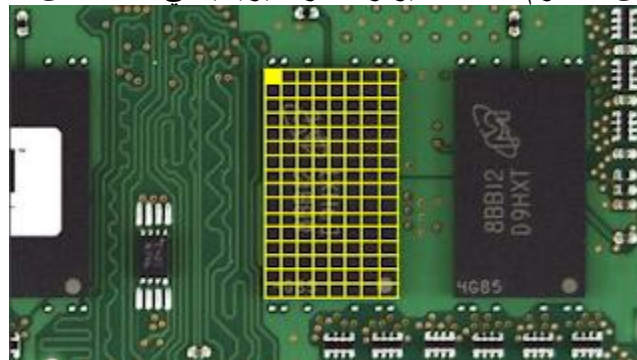


كل black chips موجود فالصورة يعبر عن عدد من bytes بالرغم من شكلها الصغير فيمكنها التعبير عن بليون من bytes لو لديك Ram 1 gigabyte على سبيل المثال وبالفعل عند تشغيل برنامج يتم حفظ المعلومات في هذه Black chips ويمكننا إعتبار أن هذه الرقائق مقسمة إلى grid كالتالي

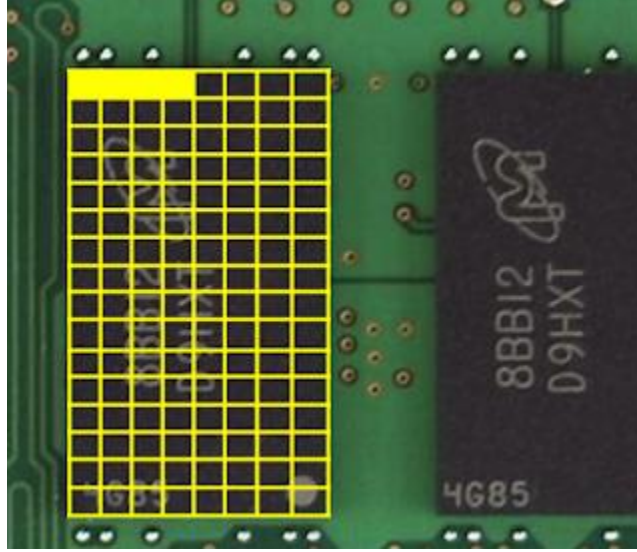


يمكننا إعتبار أن كل مربع صغير بداخل grid يعبر عن byte فيمكن داخل واحدة فقط من black chips التعبير عن بليون byte لا يهم حاليا معرفة كيفية عمل هذه الرقائق فيزيائيا ولكن المهم معرفة أن كل byte يأخذ رقم أول مربع صغير من جهة اليسار يعتبر رقم 0 ثم 1 ثم 2 وهكذا

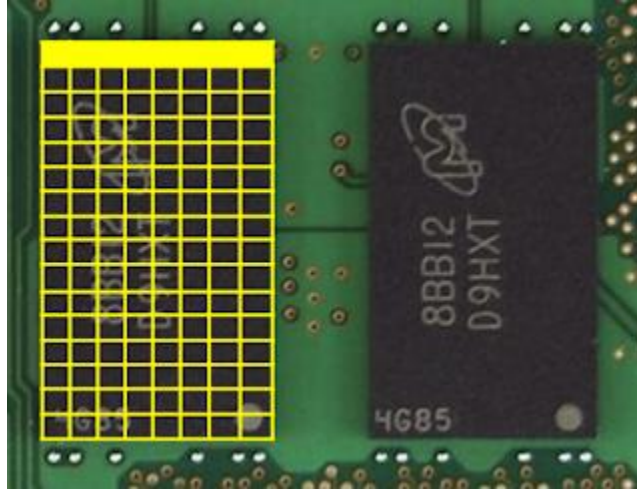
مثال وليكن لدينا data type من النوع char نريد حفظ فيها حرف واحد ومن الدروس السابقة عرفنا أنه يستخدم 1 byte من الذاكرة لذا سيكون المعلوم عند الكمبيوتر حجته فيزيائيا في هذا المكان.



على سبيل المثال لو أردنا استخدام integer فنحفظه فالذاكرة في 4 bytes ويكون فيزيائيا على الشكل التالي



ولو أردنا حفظ وليكن double يكون على النحو التالي



وفعليا بداخل كل مربع صغير يوجد 8 bits
فيزيائيا يمكن أن يكون 8 من transistors أو 8 من bulbs أي يكون بيتم التعبير عنهم فيزيائيا بأي طريقة
ولكنه يتم التعبير عن 8 bit من الأصفار والأحاد بداخل كل مربع للتعبير عن byte الواحدة وكل مربع يكون له
عنوان بترتيب العد التسلسلي من رقم صفر

Week 2 - (Arrays)

محتوى هذا الجزء:

- Arrays
- Strings
- Command line arguments
- Readability
- Encryption

Arrays

يمكنك إيجاد الأمثلة في هذا اللينك [إضغط هنا](#)
على سبيل المثال نريد إنشاء هذا البرنامج

المثال الأول:

(أنظر المثال hi.c)

```
#include <stdio.h>
4
5 int main(void)
6 {
7     char c1 = 'H';
8     char c2 = 'I';
9     char c3 = 'I';
10    printf("%c %c %c\n", c1, c2, c3);
11 }
```

Terminal

```
$ make hi
clang -fsanitize-signed-integer-overflow -fsanitize-undefined -gdb3 -O0 -std=c11 -Wall -Wshadow hi.c -lcrypt -lcs50 -ln -o hi
$ ./hi
H I I
$
```

ملحوظة هنا نستخدم ' ' لانه حرف واحد ونستخدم " " للتعبير عن string
فعليا يكونو فالذاكرة على الشكل التالي

H	I	!					
c1	c2	c3					

يمكننا رؤية الرقم الفعلي للحروف هذه بناء على جدول اسكي الذي تم شرحه سابقا كالتالي

```

hi.c
1 // Prints ASCII codes
2
3 #include <stdio.h>
4
5 int main(void)
6 {
7     char c1 = 'H';
8     char c2 = 'I';
9     char c3 = '!';
10    printf("%i %i %i\n", c1, c2, c3);
11 }

```

```

Terminal
$ make ht
clang -fsanitize=signed-integer-overflow -fsanitize=undefined -ggdb3 -O
-Wshadow ht.c -lcrypt -lcs50 -ln -o ht
$ ./ht
72 73 33
$

```

وهنا كان يجب كتابة الكود الخاص بالطباعة على الشكل التالي وهذه الطريقة تسمى (CAST) casting وهو تحويل data type إلى أخرى

```

hi.c
1 // Prints ASCII codes
2
3 #include <stdio.h>
4
5 int main(void)
6 {
7     char c1 = 'H';
8     char c2 = 'I';
9     char c3 = '!';
10    printf("%i %i %i\n", (int) c1, (int) c2, (int) c3);
11 }

```

ولكن compiler يمكنه عمل ذلك والتعرف عليه بدلا من كتابته بهذا الشكل ويكون الناتج فالذاكرة كالتالي

72	73	33					
c1	c2	c3					

والذي يكون بناء على اعداد binary كالشكل التالي

01001000	01001001	00100001					
c1	c2	c3					

المثال الثاني:

أنظر المثال Scores0

```

3 #include <cs50.h>
4 #include <stdio.h>
5
6 int main(void)
7 {
8     // Scores
9     int score1 = 72;
10    int score2 = 73;
11    int score3 = 33;
12
13    // Print average
14    printf("Average: %i\n", (score1 + score2 + score3) / 3);
15 }

```

هذا برنامج بسيط لحساب متوسط ثلاث أرقام والنتائج كالتالي

```

>_ Terminal x +
$ make scores0
clang -fsanitize-signed-integer-overflow -fsanitize-undefined -ggdb
-Wshadow scores0.c -lcrypt -lcs50 -lm -o scores0
$ ./scores0
Average: 59
$

```

ولكن هذا البرنامج تصميمه سيء لأنه يحسب متوسط ثلاث أرقام فقط ولا يمكننا إضافة رقم جديد عند الحاجة بسهولة .

وهنا أتضح أن في الذاكرة نستطيع تخزين المتغيرات واحدة تلو الأخرى أي بطريقة متتالية وفي C أيضا تكون المتغيرات واحدة تلو الأخرى في جزء من الذاكرة وهذه تسمى مصفوفة **Array** أذن Array نستخدمها لمجموعة من المتغيرات من نفس النوع ولهم نفس الاسم لذا يمكننا تعديل البرنامج السابق كالتالي لتحسينه وإستخدام المصفوفة

المثال الثالث :

أنظر المثال scores1.c

```
2
3 #include <cs50.h>
4 #include <stdio.h>
5
6 int main(void)
7 {
8     // Scores
9     int scores[3];
10    scores[0] = 72;
11    scores[1] = 73;
12    scores[2] = 33;
13
14    // Print average
15    printf("Average: %i\n", (scores[0] + scores[1] + scores[2]) / 3);
16 }
```

>_ Terminal x +

```
$ make scores1
clang -fsanitize=signed-integer-overflow -fsanitize=undefined -g -gdb3 -O
-Wshadow scores1.c -lcrypt -lcs50 -ln -o scores1
$ ./scores1
Average: 59
$
```

أنظر طريقة كتابة المتغير بداخل المصفوفة (أنظر الفيديو للشرح ملحوظة

Array بداية العد به من Zero وكتبنا الرقم 3 لتعريف طول المصفوفة وهنا نخبره أنه رقم ثابت أي ان المصفوفة بها 3 ارقام فقط
وهنا لدينا نفس المشكلة السابقة أننا قمنا بتحديد عدد القيم في المصفوفة وجعلناها رقم ثابت ولكن هنا يمكننا إستخدام مصفوفة بعدد غير معلوم لحل وتحسين البرنامج السابق كالتالي

```
6 int main(void)
7 {
8     int n = 3;
9     int scores[n];
10    scores[0] = 72;
11    scores[1] = 73;
12    scores[2] = 33;
13
14    // Print average
15    printf("Average: %i\n", (scores[0] + scores[1] + scores[2]) / n);
16 }
```

ولكن هناك ميزة في لغة C وأيضا العديد من لغات البرمجة عند إستخدام متغير في أكثر من مكان في البرنامج

كالسابق مع حالة المتغير n هنا نستخدم const للإشارة إلى أنه يجب أن يكون هو نفسه دائما في كلا المكانين ويكتب كالتالي:

المثال الرابع:

أنظر المثال Scores2.c

```
3 #include <cs50.h>
4 #include <stdio.h>
5
6 const int N = 3;
7
8 int main(void)
9 {
10     int scores[N];
11     scores[0] = 72;
12     scores[1] = 73;
13     scores[2] = 33;
14
15     printf("Average: %i\n", (scores[0] + scores[1] + scores[2]) / N);
16 }
```

>_ Terminal x +

```
$ make scores2
clang -fsanitize=signed-integer-overflow -fsanitize=undefined -ggdb3 -O0
-Wshadow scores2.c -lcrypt -lcs50 -ln -o scores2
$ ./scores2
Average: 59
$
```

هنا نستخدم كلمة Const لإخبار compiler أن قيمة المتغير N يجب ألا تتغير بواسطة البرنامج . ووفقا للاتفاقيات سنضع المتغير global أي بخارج function ونجعله Capital كما مكتوب فالمثال السابق وهذا ليس ضروريا compiler ولكنه متعارف للبشر أن هنا متغير ثابت ويكون أكثر سهولة للرؤية من البداية. يمكننا جعل البرنامج السابق يعمل اتوماتيكيا مع العدد الذي يريده المستخدم والأرقام التي يدخلها كالمثال التالي:

المثال الخامس:

أنظر المثال scores3.c

```

scores3.c x +
1 // Averages numbers using a helper function
2
3 #include <cs50.h>
4 #include <stdio.h>
5
6 int main(void)
7 {
8     // Get number of scores
9     int n = get_int("Scores: ");
10
11     // Get scores
12     int scores[n];
13     for (int i = 0; i < n; i++)
14     {
15         scores[i] = get_int("Score %i: ", i + 1);
16     }
17
18     // Print average
19     printf("Average: TODO\n");
20 }
21

```

هنا تجد أنه يطلب من المستخدم إدخال قيمة عدد النتائج لمعرفة طول المصفوفة ثم وضع for loop لإنشاء المصفوفة بالعدد الذي حدده المستخدم (أنظر الشرح فالفيديو) ويكون الناتج كالتالي

```

Terminal x +
$ make scores3
clang -fsanitize=signed-integer-overflow -Wshadow scores3.c -lcrypt -lc
$ ./scores3
Scores: 3
Score 1: 73
Score 2: 72
Score 3: 33
Average: TODO
$

```

ولكن كيف يمكننا حساب المتوسط وطباعته ديناميكيا بدون معرفة المصفوفة وعددها التي سوف يدخلها المستخدم هنا سنستخدم function مساعدة كالتالي

```

24 float average(int length, int array[])
25 {
26     int sum = 0;
27     for (int i = 0; i < length; i++)
28     {
29         sum += array[i];
30     }
31     return (float) sum / (float) length;
32 }

```

هذه function تأخذ 2 inputs وهم طول المصفوفة وقيم array
ثم تجمع قيم المصفوفة معا
ثم تقوم بإرجاع المجموع مقسوم على طول المصفوفة ولو لم نكتب float sum كما مكتوب سابقا سيكون الناتج
ارقام int فهنا نستخدم هذه الميزة من لغة C
سيكون البرنامج كامل كالتالي

```
scores3.c
1 #include <cs50.h>
2 #include <stdio.h>
3 float average(int length, int array[]);
4 int main(void)
5 {
6     // Get number of scores
7     int n = get_int("Scores: ");
8
9     // Get scores
10    int scores[n];
11    for (int i = 0; i < n; i++)
12    {
13        scores[i] = get_int("Score %i: ", i + 1);
14    }
15    // Print average
16    printf("Average: %.1f\n", average(n, scores));
17 }
18 float average(int length, int array[])
19 {
20     int sum = 0;
21     for (int i = 0; i < length; i++)
22     {
23         sum += array[i];
24     }
25     return (float) sum / (float) length;
26 }
```

وهنا ستجد في السطر الثالث قام بتعريف الكمبيوتر بأنه يوجد function بالاسفل وهذه خواصها كما شرحنا سابقا
ثم فالسطر 16 الكود يشرح طباعة رقم float ناتج عن function التي نستدعيها والتي تسمى average والتي
تأخذ قيمة n أي طول المصفوفة و scores قيم المصفوفة (أنظر الشرح بالفيديو)

ويكون الناتج كالتالي

```
>_ Terminal
$ make scores3
clang -fsanitize=signed-integer-overflow -fsanitize=
-Wshadow scores3.c -lcrypt -lcs50 -ln -o scores3
$ ./scores3
Scores: 2
Score 1: 100
Score 2: 99
Average: 99.5
$
```

ويكون شكله في الذاكرة لحفظ هذه القيم من النوع int كالتالي

72 scores[0]	73 scores[1]						
33 scores[2]							

Strings

يمكننا إعتبارها أنها array من chars وكل حرف يمكن برقم في المصفوفة وليكن لدينا string كالتالي

```
string s = "HI!";
```

هنا فعليا يتم تخزينها فالذاكرة كالتالي

H s[0]	I s[1]	! s[2]	\0 s[3]				

كل حرف يكون له رقم فالمصفوفة فعليا بعد إنتهاء string يكون هناك byte غير مرئي يستخدمه الكمبيوتر لتعريف إنتهاء هذه المصفوفة ويكون \0 أي بايت من 8 zero لذلك عند كتابة string وليكن من 8 أحرف فعليا يستخدم الكمبيوتر 9 بايت لتخزينها .

المثال السادس:

أنظر المثال names.c

```

3 #include <cs50.h>
4 #include <stdio.h>
5 #include <string.h>
6
7 int main(void)
8 {
9     // Names
10    string names[4];
11    names[0] = "EMMA";
12    names[1] = "RODRIGO";
13    names[2] = "BRIAN";
14    names[3] = "DAVID";
15
16    // Print Emma's name
17    printf("%s\n", names[0]);
18    printf("%c%c%c%c\n", names[0][0], names[0][1], names[0][2], names[0][3]);
19 }

```

هنا تكون أول printf تطبع أول string في المصفوفة
ثاني printf هنا نطلب منه طباعة كل حرف على حدى من string الأول من المصفوفة فعليا يمكن أن تكون
printf تستخدم بداخلها loop لطباعة String وعندما تصل إلى byte 0 \0 تقف عند الطباعة.
وتكون النتيجة كالتالي

```

$ make names
clang -fsanitize=signed-integer-overflow -fsanitize=shadow names.c -lcrypt -lcs50 -lm -o names
$ ./names
EMMA
EMMA
$ █

```

شكل المثال السابق في الذاكرة كالتالي

E	M	M	A	\0	R	O	D
name1					name2		
R	I	G	O	\0	B	R	I
					name3		
A	N	\0	D	A	V	I	D
			name4				
\0							
E	M	M	A	\0	R	O	D
names[0][0]	names[0][1]	names[0][2]	names[0][3]	names[0][4]	names[1][0]	names[1][1]	names[1][2]
R	I	G	O	\0	B	R	I
names[1][3]	names[1][4]	names[1][5]	names[1][6]	names[1][7]	names[2][0]	names[2][1]	names[2][2]
A	N	\0	D	A	V	I	D
names[2][3]	names[2][4]	names[2][5]	names[2][6]	names[2][7]	names[3][0]	names[3][1]	names[3][2]
\0							
names[3][3]							

في الصورة السابقة يكون أول رقم بين الأقواس يعبر عن رقم names أي رقم input وثاني رقم بين
الأقواس يعبر عن الحرف بداخل هذا input

ملحوظة

لو طلبنا طباعة names[0][4] على أنه int سيكون الناتج صفر كالتالي

```
7 int main(void)
8 {
9     // Names
10    string names[4];
11    names[0] = "EMMA";
12    names[1] = "RODRIGO";
13    names[2] = "BRIAN";
14    names[3] = "DAVID";
15
16    // Print Emma's name
17    printf("%i\n", names[0][4]);
18 }
19
20
```

>_ Terminal x +

```
$ make names
clang -fsanitize=signed-integer-overflow -Wshadow names.c -lcrypt -lcs50 -l
$ ./names
0
```

المثال السابع:

أنظر المثال string0.c

```
3 #include <cs50.h>
4 #include <stdio.h>
5
6 int main(void)
7 {
8     string s = get_string("Input: ");
9     printf("Output: ");
10    for (int i = 0; s[i] != '\0'; i++)
11    {
12        printf("%c", s[i]);
13    }
14    printf("\n");
15 }
16
```


هنا نريد طباعة string يقوم بإدخاله المستخدم لذا سنضع شرط داخل loop تقف عندما يصل إلى \0 أي إنتهاء string كما شرحنا سابقا وهنا نطلب منه طباعة string كأحرف .
ملحوظة :

هنا وضعنا ' ' مع أنهم حرفين وليس حرف واحد وينبغي كما شرحنا سابقا وضع " " ولكن هنا يمكن إعتبارها حرف واحد لأنها مثل \n تؤدي نفس المعني.
وهذه طريقة أخرى لإستخدام %c بدلا من %s وهذا سيكون الناتج

```
>_ Terminal x +
$ make string0
clang -fsanitize=signed-integer-overflow -fsanitize=undefined -gg
-Wshadow string0.c -lcrypt -lcs50 -ln -o string0
$ ./string0
Input: lara
Output: lara
```

هناك طريقة أخرى لوضع الشرط بشكل آخر بداخل loop كالتالي

المثال الثامن:

أنظر المثال string1.c

```
3 #include <cs50.h>
4 #include <stdio.h>
5 #include <string.h>
6
7 int main(void)
8 {
9     string s = get_string("Input: ");
10    printf("Output: ");
11    for (int i = 0; i < strlen(s); i++)
12    {
13        printf("%c", s[i]);
14    }
15    printf("\n");
16 }
```

هنا وضعنا الشرط $strlen(s) > i$ وهذا إختصار ل string length أي طول string ولكن لإستخدام هذه الخاصية يجب إستخدام مكتبة جديدة كما موضح في الصورة السابقة لتعريف هذه الخاصية وستجدو عند التجربة سيعطي نفس النتيجة السابقة.

ولكن يمكننا إعتبار هذا التحديث للبرنامج غير كفاً كتصميم للبرنامج لأنه في كل مرة الكمبيوتر يحتاج للسؤال عن طول string مع كتابة كل حرف في حين أنه لو ثابت يظل الكمبيوتر يسأل عن الطول ويمكننا التغلب على هذا العيب كالتالي

```

3 #include <cs50.h>
4 #include <stdio.h>
5 #include <string.h>
6
7 int main(void)
8 {
9     string s = get_string("Input: ");
10    printf("Output: ");
11    int n = strlen(s);
12    for (int i = 0; i < n; i++)
13    {
14        printf("%c", s[i]);
15    }
16    printf("\n");
17 }

```

يمكننا كتابة البرنامج بالشكل التالي

المثال التاسع:

أنظر المثال string2.c

```

3 #include <cs50.h>
4 #include <stdio.h>
5 #include <string.h>
6
7 int main(void)
8 {
9     string s = get_string("Input: ");
10    printf("Output: ");
11    for (int i = 0, n = strlen(s); i < n; i++)
12    {
13        printf("%c", s[i]);
14    }
15    printf("\n");
16 }

```

نفس البرنامج السابقة ولكن يمكننا إختصاره في سطر واحد وهنا يسأل عن طول String مرة واحدة وحين لم تتغير فلا يسأل عنها مرة أخرى.

وستجد أنه في بداية loop قام بتعريف متغيرين منهم n ووضع بها طول string وهنا فعليا يمكنه السؤال عن قيمة المتغير n في كل مرة بدون عمل الحسابات لمعرفة طول string في كل مرة.

في هذه الحالة نحتاج لمساحة من الذاكرة للمتغير n ولكن وفرنا في البرنامج الوقت لحساب طول string لكل مرة بداخل loop.

مثال آخر لكيفية كتابة الحروف uppercase

المثال العاشر:

أنظر المثال uppercase0.c

```
3 #include <cs50.h>
4 #include <stdio.h>
5 #include <string.h>
6
7 int main(void)
8 {
9     string s = get_string("Before: ");
10    printf("After: ");
11    for (int i = 0, n = strlen(s); i < n; i++)
12    {
13        if (s[i] >= 'a' && s[i] <= 'z')
14        {
15            printf("%c", s[i] - 32);
16        }
17        else
18        {
19            printf("%c", s[i]);
20        }
21    }
22    printf("\n");
23 }
```

أولا حصلنا على string وسألنا لو الأحرف lowercase حولها إلى uppercase وغير ذلك أطلعها كما هي. كيف يمكننا تحويل الأحرف من lowercase إلى uppercase بناء على الجدول التالي وكما شرحناه سابقا نظام ASCII يمكننا الحصول عليه من

<https://www.asciichart.com> موقع

0	NUL	16	DEL	32	SP	48	0	64	@	80	P	96	.	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	\$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[107	k	123	[
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124]
13	CR	29	GS	45	-	61	=	77	M	93]	109	m	125]
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

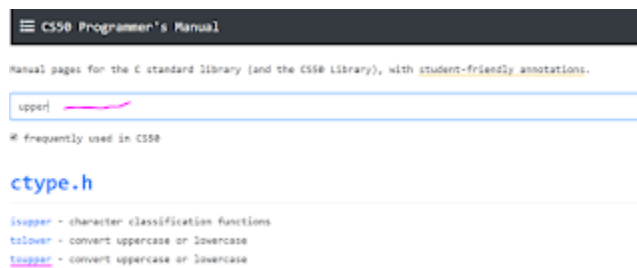
ستجد أنه بين كل حرف small و capital لنفس الحرف فرق القيمة 32 لذلك فالمثال السابق طلبنا لو الحرف small أطلع منه 32 حتى تحصل على الحرف capital

وسيكون الناتج كالتالي

```
>_ Terminal x +
$ make uppercase0
clang -fsanitize=signed-integer-overflow -f
-Wshadow uppercase0.c -lcrypt -lcs50 -
$ ./uppercase0
Before: rasha
After:  RASHA
$
```

manual pages

في المثال السابق يوجد التفاصيل الدقيقة التي يمكنني الإستغناء عنها مثل فرق القيم بين الأحرف من جدول aschii هنا نلجأ إلى manual pages أو documentation يمكننا الإستعانة بصفحات مثل الصفحة التالية الخاصة بكورس cs50 في [/https://man.cs50.io](https://man.cs50.io) ستجد الصفحة التالية



يمكنك البحث عن كلمة upper لأختيار نوع function التي تريدها وستجد نتيجة البحث كالتالي بعد الضغط على toupper

```
#include <ctype.h>

int toupper(int c);
int tolower(int c);

int toupper_l(int c, locale_t locale);
int tolower_l(int c, locale_t locale);
```

هنا يخبرك عن اسم المتغير toupper ونوعه int واسم المكتبة ctype.h التي يجب إستدعاءها دعونا نجرب المثال السابق مرة أخرى بالدالة الجديدة أنظر المثال uppercase2.c

```

3 #include <cs50.h>
4 #include <ctype.h>
5 #include <stdio.h>
6 #include <string.h>
7
8 int main(void)
9 {
10     string s = get_string("Before: ");
11     printf("After: ");
12     for (int i = 0, n = strlen(s); i < n; i++)
13     {
14         printf("%c", toupper(s[i]));
15     }
16     printf("\n");
17 }

```

وسيكون الناتج كالتالي

```

>_ Terminal x +
$ make uppercase0
clang -fsanitize=signed-integer-overflow -Wshadow uppercase0.c -lcrypt -lcs
$ ./uppercase0
Before: lara
After: LARA
$

```

ملحوظة:

يمكنك استخدام أى من functions الموجودة في problem sets حتى ولم نستخدمها في المحاضرة

Command line arguments

هي ميزة أخرى يمكننا استخدامها في C مع main function حين نريد إدخال قيمة من user ولكن عن طريق command line
 مثال سابق على command line argument حين استخدمنا -o مع الأمر clang لتغيير اسم ملف output

ويتم كتابتها كالتالي

```

#include <stdio.h>

int main(int argc, string argv[])
{
    ...
}

```

argc, argv هما متغيرين يتم إدخالهم في هذه الحالة لل main function اول متغير هو int argc وهو متغير لحساب عدد arguments و argv هو string of array به arguments . أنظر الشرح بالفيديو

ملحوظة [0] argv هو اسم الملف وليكن مثلاً ./hello يعتبر هو اول اسم في المصفوفة
انظر الشرح بالفيديو للتوضيح

المثال الحادي عشر:

أنظر مثال argv.c

```
3 #include <cs50.h>
4 #include <stdio.h>
5
6 int main(int argc, string argv[])
7 {
8     if (argc == 2)
9     {
10         printf("hello, %s\n", argv[1]);
11     }
12     else
13     {
14         printf("hello, world\n");
15     }
16 }
```

في المثال السابق يوجد شرط لو عدد arg المدخلة من user في command line بيساوي اثنين أطبع كلمة hello مع الاسم الموجود في المصفوفة الثاني [1] argv أن لم يتحقق الشرط قم بطباعة كلمة hello world سيكون الناتج كالتالي

```
>_ Terminal x +
$ make argv
clang -fsanitize=signed-integer-overflow -fsanitize=undefined -g
-Wshadow argv.c -lcrypt -lcs50 -ln -o argv
$ ./argv
hello, world
$ ./argv rasha
hello, rasha
$
```

ستجد أنه حين كتبت في command line الامر ./argv فقط بمعنى أنه لم يتحقق الشرط وان عدد input يساوي واحد سيكون الناتج hello world على العكس إن أدخلنا أسم كما موضح بالصورة.

ملحوظة:

int main(int argc, string argv[])

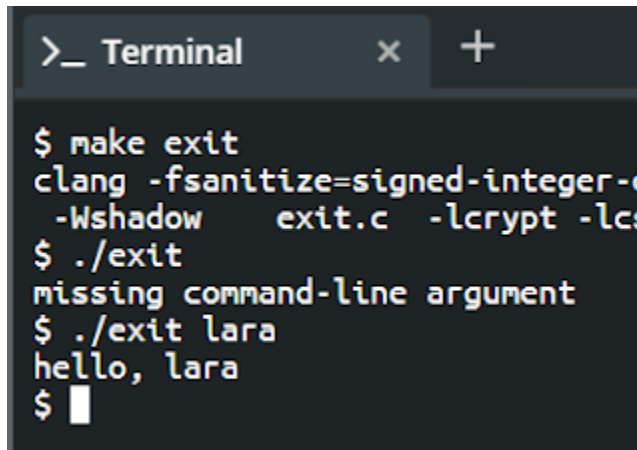
int تعني output وهنا يمكننا تعريف إذا يوجد خطأ في البرنامج أم لا فالمتعارف في هذه الحالة لو الرقم صفر فإنه لا يوجد أخطاء ويمكننا إنشاء برنامج لتغيير لإعطاء قيمة مختلفة كالتالي

المثال الثاني عشر:

أنظر المثال التالي exit.c

```
2
3 #include <cs50.h>
4 #include <stdio.h>
5
6 int main(int argc, string argv[])
7 {
8     if (argc != 2)
9     {
10         printf("missing command-line argument\n");
11         return 1;
12     }
13     printf("hello, %s\n", argv[1]);
14     return 0;
15 }
```

القيمة المرجعة في هذه الحالة تسمى exit code وهي غير مرئية لمستخدم ولكنها كإشارة للكمبيوتر بأنه يوجد خطأ.
ويكون الناتج كالتالي



```
>_ Terminal x +
$ make exit
clang -fsanitize=signed-integer-compare -Wshadow exit.c -lcrypt -lcs50
$ ./exit
missing command-line argument
$ ./exit lara
hello, lara
$
```

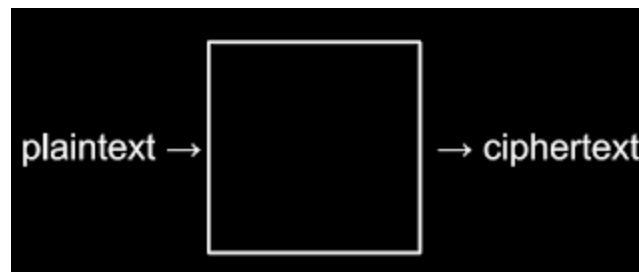
Readability

يمكننا استخدام ما سبق لحل العديد من المشاكل على سبيل المثال القراءة.
وبعد أن عرفنا كيفية التعامل مع strings في برامجنا , يمكننا تحليل الجمل لمستوى قابليتها للقراءة بناءً على طول وتعقيد الكلمات والجمل

Encryption

إذا أردنا إرسال رسالة نصية إلى شخص ما فأنا نريد تشفيرها encrypt أو بطريقة ما نريد التشويش على هذه الرسالة حتى يكون من الصعب قراءتها من الآخرين
ملحوظة :

مصطلح cryptography يعني the art of scrambling information to keep it private ويكون له input وهو الرسالة التي نريد إرسالها وفي هذه الحالة تسمى plaintext و output هو الرسالة بعد التشفير وتسمى ciphertext وما بالداخل هو algorithm المتبع لتشفير هذه الرسالة.



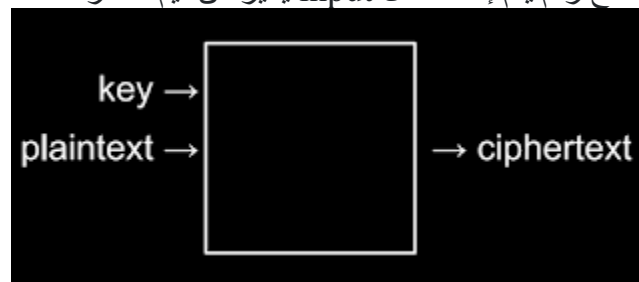
مثال وليكن نريد إرسال رسالة

H I !

سيتم تحويلها عن طريق نظام Ascii بالارقام إلى

72 73 33

ولكن من السهولة لأي شخص إعادة تحويل الارقام مرة أخرى ومعرفة ماهي الرسالة و هنا نحتاج إلى key او رقم لا يعرفه غير المرسل والمستلم فقط هذا المفتاح رقم يتم إضافته لل input يغير من قيم الأحرف



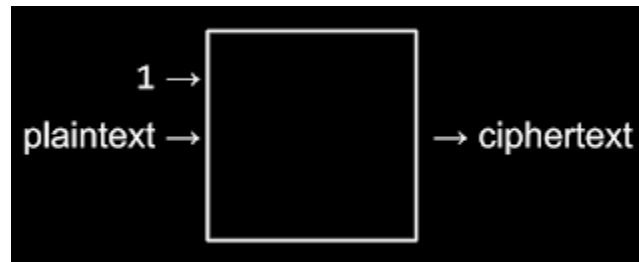
على سبيل المثال
لو أردنا إرسال كلمة

I L O V E Y O U

عن طريق ascii سيكون كالتالي

73 76 79 86 69 89 79 85

وليكن key رقم 1



وبناء عليه سيتغير إلى

74 77 80 87 70 90 80 86

وستحول الرسالة إلى

J M P W F Z P V

ولفك هذه الشفرة يجب معرفة key

وذلك بناء على algorithm تحول بين الرسالة المرسله والرسالة بعد التشفير والعكس.