المختصر في
البرمجة المتقدمة في السي تحت نظام لينكس

بسم الله الرحمن الرحيم

مقدمة الكتاب :

الحمد الله الذي شاء ان اكتب هذا الكتيب الذي اطمع ان يكون قربانا لله و ينفع كل طالب علم في الامة.

ام بعد

دون اطالة في الكلام لاني ساحول تطبيق "خير الكلام ما قل ودل" وعدم تكرار ماسبق ان كتب لذالك في بعض الفصول سأضع اسماء لكتب ووثائق وحتى مواقع ان شاء الله فالداعي لكتابة الكتاب عدة اسباب وساذكر بعضها ثم حلول لبعض المشاكل في رسالة الكتاب ثم كتب ينصح ها.

اولا :

ان المكتبة العربية تعاني من نقص كبير في الكتب المؤلفة او المترجمة في جميع العلوم وخاصة مجال المعلوماتية .

ثانيا :

ان المكتبة العربية تعج وتعاني من كثرة الكتب التي تعتبر عبئ عليها من حيث المحتوى المكرر ولا جديد فيها.

ثالثا :

كتاب الاروع بين كل الكتب في تخصصه وهو " **كتاب الشامل في السي** " الذي لم اجد منافس لهذا الكتاب سو كتابين في المكتب العربية وهما " **الشامل في اللينكس** " و " **الاكسير في سي بلس** " فلم اجد افضل منهم في السي هم اروع ما كتب وهي كتب جيدة للمبتدئين .

فبعد انهائي لهذا الكتاب اردة التعلم المزيد عن اللغة فبحثت في المكتبة العربية والمواقع العربية عن المستوى التالي للغة فلم اجد ولاكتاب الا بعض المشاركات في الحقيقة كانت المشاركة الواحدة منها تنسخ في 20 موقع بين منتدى و مدونة ..... على الاقل كاننا لا نجيد سو النسخ للاسف .

تقديم الكتاب :

هذا الكتاب هو تحت رخصة " وقف " لمراجعة الرخصة ارجو مراجعة الموقع الرسمي لها " أعجوبة".

لقد كانت لي محاولات ولم اكمل والسبب هو اني لم اكن في حاجة لكي اتمرن على لوحة المفاتيح و لم يكن لي واجب لكي الخص او اعيد صياغة ما كتب

ففكرة في جمع بعض المقالات والمشاركة وترجمة بعض الوثائق مع بعض اجتهاد الشخصي تحت كتيب يكو باب للمستوى المتقدم للغة و مكملا لكتاب الاول "الشامل في السي" واضافة جديدة لمكتبتنا و للامة وارجو من الله القبول.

سيكون هذا الكتاب موجه لذوي المستوى المتوسط في اللغة السي نحو المستوى المتقدم ان شاء الله وثانيا ساحول تبسيط الامور قدر المستطاع مع الامثلة بصورة ملخصة ومباشرة لاني لاحبذ التعقيد واستعمال اساليب اللغة معقدة ، هدف الكتاب وضع امثلة برمجية تكون عامة بشرح مختصر ومباشر شرح الامور الجديدة والهامة.

ملاحظة الكتاب :

لكل شيء وقت فلا تقلق ولا تحزن وبعضها يحتاج الى الوقت ليفهم.

اغلب فصول الكتاب ليست متسلسلة فيمكن كسر التسلسل المتبع ولكنه لايستحسن لقليل الخبرة.

رسالة الكتاب :

هذه الرسالة عبارة عن نظرة شخصية لحل نقص المعلومة في كتبنا وهي عبارة عن نقاط :

1- تعلم بلغة الام :

العربية لغة القران اجمل لغة لان هناك احصائية تقول التعلم بلغة تكون نسبة التعلم عالية ونسبة الفهم قد تصل الى 80% بينما عكس ذلك بلغة غريبة .

2- الترجمة والترجمة ثم الترجمة :

كل تغير ايجابي عبر التاريخ وكل ثورة ونهظة في امة ما كان نشاط حركة الترجمة الى لغة الام.

3- التاليف :

لانريد كم في كتب بل نريد كم في المعلومة .

4- افضل طريق لتعلم :

ابداء مشروع محاول تطبيق كل ماتتعلمه او علم واشرح -شارك الاخرين - ما تعلمت.

الفصل الأول :
تحويل بين الأنواع

مقدمة:

في الحقيقة هذا الفصل يمكن القول انه خدع وتقنيات في لغة السي و يمكن القول ان اي من ما سيرد في هذا الفصل ستحتاج اليه خلال مشواك مع السي فحيث يمكن اعتباره هدية الكتاب لان حاولت فيه جمع كل الطرق والتقنيات الشائعة وغيرها .

عداة أثناء كتابة برنامج ما قد تضطر ألى تغير نوع متغير للاستنفاد منه في بقية البرنامج لذ سنتعلم طرق تحويل بين الأنواع وبعدة طرق مختلفة والموضوع سهل إن شاء الله .

حيث إن كل ما سيقدم ستجد إن كثير من البرامج تعتمد عليه في برمجتها وخاصة في مجال برمجة الشبكات تعدد المسالك وغيرها- وسنخصص دروس في هذه المجالات إن شاء الله

تحويل بالدوال القياسية :

هي دوال موجودة مسبقا في مكتبات السي القياسية حيث توجد في <stdio.h> ويمكن الاطلاع على معلومات اكثر وذلك عن طريق الطرفية :

```
man atoi
........
man atol
........
man atof
........
```

حيث ستجد معلومات كثيرة عنها

```
(1)-int atoi(const char*  c);  int (int)strtol(str0,(char**)NULL,10);
(2)-long atol(const char*  c);   long strtol(str0,(char**)NULL,10);
(3)-double atof(const char*  c);   double strtod(str0,(char**)NULL);
```

1 - تقوم بتحويل من سلسلة حرفية إلى عدد صحيح من نوع int

2 - تقوم بتحويل من سلسلة حرفية إلى عدد صحيح من نوع long int

3 - تقوم بتحويل من سلسلة حرفية إلى عدد طبيعي من نوع double

1 -تحويل من سلسلة حرفية إلى عدد صحيح من نوع int:

```
#include<stdio.h>
#include<stdlib.h>
main (int argc, char *argv[])
{
	char *str0="epidemic";
	printf("\nstr0 = %s\n",str0);

	int in0;
	in0=atoi(str0);
	printf("str0 ---> int0 = %d\n\n",in0);

	char *str1="123456";
	printf("str1 = %s\n",str1);

	int in1;
	in1=atoi(str1);
	printf("str1 ---> int1 = %d\n\n",in1);

	char *str2="123.456";
	printf("str2 = %s\n",str2);

	int in2;
	in2=atoi(str2);
	printf("str2 ---> int2 = %d\n\n",in2);

	return 0;
}
```

إخراج البرنامج

```
[aye7@localhost ~]$ gcc -o cvrt1 cvrt1.c
[aye7@localhost ~]$ ./cvrt1

str0 = epidemic
str0 ---> int0 = 0

str1 = 123456
str1 ---> int1 = 123456

str2 = 123.456
str2 ---> int2 = 123
```

2 - تحويل من سلسلة حرفية إلى عدد طبيعي من نوع double :

```
#include<stdio.h>
#include<stdlib.h>
main (int argc, char *argv[])
{
	char *str0="epidemic";
	printf("\nstr0 = %s\n",str0);

	double in0;
	in0=atof(str0);
	printf("str0 ---> double0 = %f\n\n",in0);

	char *str1="123456";
	printf("str1 = %s\n",str1);

	double in1;
	in1=atof(str1);
	printf("str1 ---> double1 = %f\n\n",in1);

	char *str2="123.456";
	printf("str2 = %s\n",str2);

	double in2;
	in2=atof(str2);
	printf("str2 ---> double2 = %f\n\n",in2);
	return 0;
```

```
}
```

إخراج البرنامج

```
[aye7@localhost ~]$ gcc -o cvrt2 cvrt2.c
[aye7@localhost ~]$ ./cvrt2

str0 = epidemic
str0 ---> double0 = 0.000000

str1 = 123456
str1 ---> double1 = 123456.000000

str2 = 123.456
str2 ---> double2 = 123.456000
```

الشرح:

البرنامج الأول والثاني عبار عن كفية استعمال دالة التحويل من نوع سلسلة حرفية الى عدد صحيح و عدد طبيعي أم باقي ملاحظ في إخراج البرنامج كل شيئ واضح بعد تشعيل البرنامج ولا يحتاج الى شرح مطول.

هناك دوال كثيرة ولكن اخترت اكثر استعمالا وتداولا .

تحويل القصري Types Casting :

وهو سهل وبسيط تعتمد عليه كل برامج وحتى ألدوال السابقة وعملية التحويل هذه هي عملية مؤقتة، و تستخدم في الحالات التي تملك فيها بيانات ضمن نوع من المتغيرات، و أنت تريد تطبيق عمليات على هذه البيانات. العمليات تستلزم أن تكون هذه البيانات منظمة و معبر عنها بطريقة غير تلك التي يؤمنها النوع الحالي للمتغير .

تتبع عملية التحويل الطريقة الهجائية التالية :

(type-name) cast-expression

```
#include <stdio.h>

int main()
{
	float a=2;
	float b=5;
	float c =a/b;		/*	2/5=0.4	*/
	printf("c = %f \n",c);
	printf("(int)c = %i\n",(int)c);
	return 0;
}
```

```
[aye7@localhost ~]$ gcc -o enn env.c
[aye7@localhost ~]$ ./enn
c = 0.400000
(int)c = 0
```

مثال:

في هذا المثال سنعرف حل المشكل التالي:

في بعض المرات تريد ان تمرر ثلاث او اثنين من الوسائط مختلفة لدالة ما، فوجدة ان الدالة لا تستقبل إلا وسيط واحد ومن نوع محدد فما الحل تابع المثال

```
#include <unistd.h>
#include<stdio.h>
#include<errno.h>
#include<stdlib.h>
#include<pthread.h>
int x =0;
struct struct_arg
      {
            int val;
            char ch;
            char *str;
      } ;
      typedef struct struct_arg argument ;

void *mythread(void* arg)
{
      argument *arg1=(argument*)arg;
      printf("thread1 :: %d\n");
      printf("\t val  :: %d\n",arg1->val);
      printf("\t str  :: %s\n",arg1->str);
      printf("\t char :: %c\n",arg1->ch);
      return NULL;
}
int main (int argc, char *argv[])
{
      pthread_t thread1;

      argument argt;
      argt.val=2011;
      argt.ch='c';
      argt.str="Epidemic";
      if(pthread_create(&thread1,NULL,mythread,(void*)&argt)!=0){
      perror("pthread_create");
      exit(-1);
      }
      if(pthread_join(thread1,NULL)!=0){
      perror("pthread_join ");
      exit(-1);
      }

      return 0;
}
```

out:

```
[aye7@localhost ~]$ ./cast
thread1 :: 12046472
       val  :: 2011
       str  :: Epidemic
       char :: c
```

الشرح:

في هذا المثال استعملنا دوال الخيوط - thread - دوال المستوى العالي سيتم شرحها أجلا أما ما يهم هو

ال- struct :- عبار عن تركيب يحوي أنواع مختلفة يمكن القول ان دوره هو عبارة حاوية لجمع وضغط ما أريد تمريره لدالة المهام

الدالة - pthread_create-: لبناء الخيوط لها أربعة وسائط وسيلة ربط البرنامج الرئيسي بالدالة المهام وتمرر لها وسيط دائما يكون من نوع - VOID- وان لم يكن يجب عليك ان تقوم بتحويل القصري إليه كما يوضح الشكل في الوسيط الأخير للدالة

والدالة – mythread- : هي داله المهام و الفعلية للخيط حيث كل ما تود فعله يكون داخلها اي هي نواة الخيوط وهي دائما من نوع - VOID- وتستقبل وسيط واحد من نوع - VOID- وكما نلاحظ انه تم تمرير له وسيط به ثلاث وسائط دون الإخلال بها ثم بعد ذالك تم فتح الضغط عن تلك الوسائط بتحويل القصري

تحويل بواسطة عملية الحسابية:

وهي عملية بسيطة يمكن إن تقول أنها حلية جميلة لكن لها شروط وهي ان النوع الذي تريد يكون ذا منزلة واحد اي لا يجب ان يكون سلسلة حرفية او عدد تبين ذالك في هذا المثال

```
#include <stdio.h>

int main()
{
	char id;
	id='9';  id="1235";
	int var=id-'0';
	printf("var :%i\n",var);

	int id1=5;  id1= 123;
	char var1=id1+'0';
	printf("var1 : %c\n",var1);

	return 0;
}
```

```
[aye7@localhost ~]$ ./enn
var : 9
var1 : 5
```

تحويل من نوع الرقمي الى نوع الحرفي :

في هذا التحويل سنستعمل دالة جميلة وهي *sprintf()* ففي التحويل السابق هناك شرط ان يكون ذا رتبة واحدة اي ان يكون محصور الرقم مراد تحويله بين صفر والتسعة لا غير ولحل هذا المشكل نستعمل هذه الدالة

sprintf(char , const char* , ...)*

مثال

```
/*
  OS : Win , UNIX
*/
#include <stdio.h>
int main()
{
	char txt[10];
	int nbr=50;

	printf("nbr :%d\n",nbr);
	sprintf(txt,"%d",nbr);
	printf("txt :%s\n",txt);
	return 0;
}
```

```
[aye7@localhost ~]$ ./enn
nbr :50
txt :50
```

شرح نوع void :

هو نوع لا يمكن الإعلان عن إلا باستعمال المؤشر وذلك لانه ليس له حجم كي يتم إسناد له قيم هو متغير لين يمكن استعماله في حالات كثيرة والسر في ليونته هو حجمه فهو كما قولنا له حجم 0 -صفر – كما يمكن ان يمكن له حجم النوع المسند له وسوف نرى امثلة في كيفية معرف الاحجام الانوع في التالي:

```
#include <stdio.h>
int main()
{
	void *p_void;

	p_void="fffff";
	printf("void <-- string : %s\n",p_void);

	p_void=20;
```

```
        printf("void <-- int    : %i\n",p_void);

        p_void='h';
        printf("void <-- char   : %c\n",p_void);

        return 0;
}
```

```
[aye7@localhost ~]$ ./enn
void <-- string : fffff
void <-- int    : 20
void <-- char   : h
```

كما تلاحظ أن هذا المتغير له القدرة على التعامل واستقبال كل الأنواع

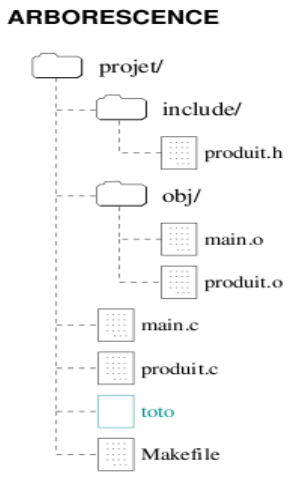
لقد تم إتمام هذا الجزء الذي هو تمهيد لبقية الدروس حيث سترتكز كل برامج القادمة بكل ما حاولنا إيصاله في هذا الجزء .

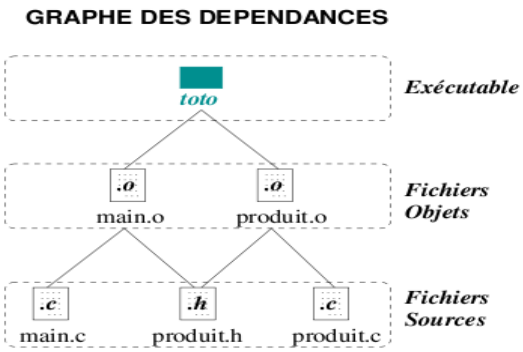
الفصل الثاني:
كتابة ملفات التركيب والتصنيف لبرنامجك

المقدمة:

عندما كتابتك لبرنامج ما فانت دائما بحاجة لتجريبه ولكي تفعل عليك ترجمته أي تصنيفه - COMPILATION -
ولكن أمر سهل مع برنامج مكون من ملف مصدر واحد أما إذا كان مشروع مكون مثلا من 10 ملفات على الأقل فما العمل هذا من جهة.
لنفرض مثلا انك تنجح دائما في ذالك فهل غيرك ينجح ؟
لان عندما تكتب شيء فانك تكتبه على أساس تشارك به الآخرين ويطلعوك على رأى كل منهم أو لينتفع به غيرك
الحل هو في برنامج - Make – وهي الاداة التي تستعمل لتركيب برامج اللينكس التي تكون عبارة عن حزمة مصدرية مم يمكننا من استعمال هذه الاداة
ويكون استعمالها بواسطة كتابة سكريت يكون دائما تحت اسم - makefile – وله نحو خاص به وبعد ذالك كل ما عليك فعله هو فتح الطرفية ثم اكتب
[aye7@localhost ~]$ make

نفرض ان ملفات المشروع في دليل المنزل والا كان عليك تتبع المسار
في البداية نفرض ان لنا مشروعين المشروع الاول مكون من ملف واحد
اما الثاني من ثلاث ملفات ونجرب الطريقتين معا عليها .
تقديم المشروع الأول: اي كود مصدري مكتوب في ملف واحد تحت اسم مثلا - main.c - .
تقديم المشروع الثاني : كود مصدري يكون مقسم كالتالي :





الملف الأول: تحت اسم - mian.c -

الملف الثاني : تحت اسم - file.h -

الملف الثالث: تحت اسم - file.cpp -

أولا: التصنيف العادي بالطريقة التقليدية

المشروع الأول:

```
[aye7@localhost ~]gcc -o nameProg main.c
[aye7@localhost ~]./nameProg
```

المشروع الثاني:

```
[aye7@localhost ~]$g++ -c file.cpp
[aye7@localhost ~]$gcc -c main.c
[aye7@localhost ~]$g++ -o nameProg main.o file.o
[aye7@localhost ~]$./nameProg
```

ثانيا: بواسطة السكريبت

المشروع الاول:

محتوى السكريبت

```
# this Commons
nameProg: main.c
	gcc -o nameProg main.c
#test u'r prog
test:
	./nameProg
```

- الانتباه للفراغ فهو مهم
- يمكن استعمال المتغيرات على النحو التالي :

```
    GCC  = gcc
-   MAIN = main.c
-   OBJ  = main.o
    $(MAIN):
    	$(VAR) -c $(OBJ)
```

-يمكن استعمل اوامر الشل والطرفية وغيرها

وفائدة المتغيرات هي مثلا لو غيرة اسم ملف في مشروعك فكل ماعليك فعله هو في المثال التالي دون العبث في السكريبت التصنيف، هذا من جهة اويمكن تغير مكان تثبيت برنامج عبر المتغيرات وغيرها من الخيارات .

```
[aye7@localhost ~]$ Make GCC= g++  MAIN=main.cpp
```

المشروع الثاني:

```
# this Commons
nameProg:
	g++ -o nameProg main.o file.o
file.o:
	g++ -c file.cpp
main.o:
	gcc -c main.c
test:
	./nameProg
```

ويمكن

```
# this Commons
PRO     = nameProg
GPP     = g++
GCC     = gcc
MAINOBJ = main.o
FILEOBJ = file.o
$(PRO):
	$(GCC) -o $(PRO) $(MAINOBJ) $(FILEOBJ)
$(FILEOBJ) :
	$(GPP)-c file.cpp
$(MAINOBJ) :
	$(GCC) -c main.c
test:
	./$(PRO)
```

وهناك الكثير الا اني اختصر لانه موضوع كبير ومتشعب ولكي تعرف عنه المزيد كل ماعليك هو قراءة الملفات التي تأتي مع تطبيقات اللينكس .

الفصل الثاني :

برمجة وكتابة برامج ذات نوعية في لينكس

1. مفهوم برمجة وكتابة برامج ذات نوعية في لينكس .
2. شرح كتابة برنامج باستخدام مكتبة <getopt.h>.

1.مفهوم برمجة وكتابة برامج ذات نوعية في لينكس :

هي عبارة خطة اتفق عليها المبرمجون لبناء برامج على منصة للينكس فمن المعلوم عند بناء اي برنامج فان برنامجك لا يقوم بعملية واحدة وانما عدة عملية مبرمجة ولكن للوصول لها يجب بناء طريق للوصول لكل عملية أو مهمة لذالك جاء الحل ببناء مكتبة خاصة للتعامل معها.

للتوضيح مثلا ls لعرض الملفات توجد به اختيارات أخرى للوصل إليها نضيف للأمر السابق إشارة ناقص ثم رمز الاختيار

-

```
[aye7@localhost ~]$ ls /
bin  dev  home  lost+found  mnt  proc  sbin   srv  tmp  var
boot  etc  lib  media      opt  root  selinux  sys  usr
```

```
[aye7@localhost ~]$ ls / -l
total 100
dr-xr-xr-x   2 root root  4096 2011-01-12 03:28 bin
dr-xr-xr-x   4 root root  4096 2010-12-29 21:14 boot
drwxr-xr-x  20 root root  3980 2011-01-12 15:18 dev
drwxr-xr-x 126 root root 12288 2011-01-12 15:20 etc
drwxr-xr-x   3 root root  4096 2010-12-29 21:16 home
.....
.....
```

اضن أن الأمر قد اتضح

شرح كتابة برنامج باستخدام مكتبة <getopt.h> :

هي مكتبة تم بنائها تسهيل التعامل arguments لبرنامج الذي انت بصدد برمجته لنطلع عليها المهم فيها توجد في المسار :

/usr/include/getopt.h

```
[aye7@localhost ~]$ cat/usr/include/getopt.h
```

or

```
[aye7@localhost ~]$ gedit /usr/include/getopt.h
```

اول شيئ شفو هذا الكود المهم :

```
1.struct option
2.{
3.  const char *name;
4.  int has_arg;
5.  int *flag;
6.  int val;
7.};
```

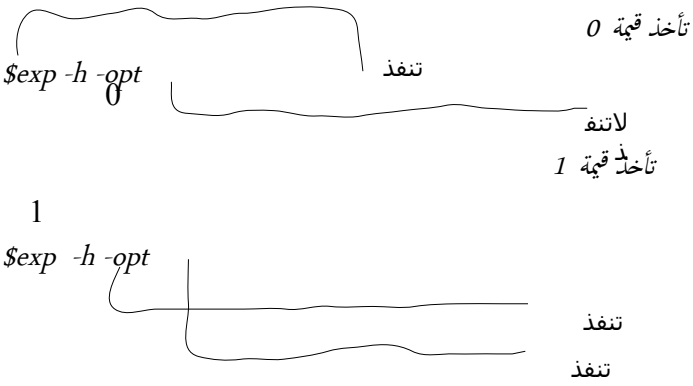
شرح استعمال الكود:

```
const struct options* long_options[]={
                                      {"help",0,NULL,'h'},
                                      {NULL,0,NULL,0}
                                      };
```

تم إعلان عن جدول من الاختيارات حيث:

الحقل الأول : اسم الكامل للاختيار أو المهمة option

الحقل الثاني :



الحقل الثالث : تأخذ NULL

الحقل الرابع :اختصار للاسم الأول

ثاني شيئ شفو هذا الدالة المهمة :

هذه دالة الربط بين الكود السابق بعض المتغيرات التي خط الغليظ و لون

```
extern int getopt_long (int ____argc, char *const *____argv,
                        const char *___shortopts,
                        const struct option *___longopts, int *___longind)
```

شرح استخدامها وطريقة الاستدعاء:

```
#include<stdio.h>
#include<stdlib.h>
#include<getopt.h>

const char* program_name;

void print_usage(FILE* stream,int exit_code)
{
fprintf(stream,"Utilisation :%s options [fichier_entree ..]\n",program_name);
fprintf(stream,
                        "-h      -- help      affiche ce message .\n"
                        "-a      -- about      About.\n");
exit(exit_code);
}

main (int argc, char *argv[])
{
        int nxt_opts;
        const char* const short_options="ha";
        const struct options* long_options[]={
                                        {"help",0,NULL,'h'},
                                        {"about",0,NULL,'a'},
```

```
                                                {NULL,0,NULL,0}
                                               };
        program_name=argv[0];
        const char* output_filename=NULL;


do
{
        nxt_opts=getopt_long(argc,argv,short_options,long_options,NULL);
                switch(nxt_opts)
        {
                case 'h':print_usage(stdout,0);
                break;
                case 'a':printf("the :%s \naye7 <Aye7_2020@hotmail.com>\n",program_name);
                break;
                case'?':print_usage(stderr,1);
                break;
                case -1:
                break;
                default:
                abort();
        }
} while (nxt_opts!=-1);

        return 0;
}
```

ملاحظة:

لاحقا سيتم شرح ما يلي

```
FILE* stream, stderr, stdout
```

الفصل الثالث :

برمجة وكتابة برامج تتعامل مع بيئة النظام تشغيل

1. *مفهوم برمجة وكتابة برامج تتعامل مع بيئة النظام تشغيل*
2. *أمثله للتطبيق*

1.مفهوم برمجة وكتابة برامج تتعامل مع بيئة النظام تشغيل:

في هذا الموضوع ستصبح قادرا على بناء برامج قوية لها قدرة على الاندماج مع نظام التشعيل من حيث تبادل معلومات بين تطبيقك ونظام التشغيل الذي تعمل عليه وبعض المتغيرات الموجودة في نظامك نأخذ على سبيل المثال الأمر في الطرفية ثم محاولة برمجة ذالك

نأخذ التالي :

```
[aye7@localhost ~]$ echo $HOME
/home/aye7
[aye7@localhost ~]$ echo $USER
aye7
```

فالمتغيرات في بيئة ألللينكس تكتب بشكل التالي :

$VAR..

```
[aye7@localhost ~]$export VAR=.....
[aye7@localhost ~]$echo $VAR
[aye7@localhost ~]$.....
```

أما بنسبة بال سي فيكون الأمر كالتالي:

```
#include<stdio.h>

main (int argc, char *argv[])
{
        printf("$USER = %s\n",getenv("USER"));
        printf("$HOME = %s\n",getenv("HOME"));
        return 0;
}
```

فقد تم الاستعانة بالدالة getenv التي تقوم باستدعاء المتغيرات من النظام

```
[aye7@localhost ~]$ ./enn
$USER = aye7
$HOME = /home/aye7
```

مثال أخر لمعرفة كل متغيرات النظام :

```
#include<stdio.h>
extern char **environ;
main (int argc, char *argv[])
{
        char **var;
        for (var = environ; *var != NULL; ++var)
        {
                printf("%s\n",*var);
        }
```

```
        return 0;
}
```

الإخراج:

```
ORBIT_SOCKETDIR=/tmp/orbit-aye7
HOSTNAME=localhost.localdomain
GPG_AGENT_INFO=/tmp/seahorse-XzpVry/S.gpg-agent:1606:1;
TERM=xterm
SHELL=/bin/bash
XDG_SESSION_COOKIE=456b4822975f096f94a8f7ef0000000a-1294841917.495671-608830466
HISTSIZE=50000
GTK_RC_FILES=/etc/gtk/gtkrc:/home/aye7/.gtkrc-1.2-gnome2
WINDOWID=88080387
GNOME_KEYRING_CONTROL=/tmp/keyring-mDBQ9x
QTDIR=/usr/lib/qt-3.3
QTINC=/usr/lib/qt-3.3/include
USER=aye7
.....
.....
.....
.....
```

الفصل الخامس:

التعابير القياسية أو العادية Regular Expretions

مقدمة :

سنرى اليوم درس جديد وجميل و هو حول التعابير القياسية ففي هذا الكتاب لا اريد التركيز على تاريخ كل تقنية لان ذالك قد نكون خرجنا عن فكرة الكتاب وهو كيفة الاستعمال لا غير و وهذاه التقنية جميلة ومفيدة جيدا وسنرى ذالك سناخذ في هذا الفصل الاساسيات ثم امثلة وبعد ذالك امثلة برمجية مع لغة السي .

مفهوم التعابير القياسية regex :

يمكن تلخصيها بان اول ضهور لها كان في انظمة اليونكس ثم بعد ذالك انتشرت في كل الانضمة و هي حاليا توجود في كل لغات البرمجة كما يمن استعمالها في الطرفية ، وهي من أقوى الطرق المستخدمة للتعامل مع النصوص في مجال البحث و المقارنة و الابدال حيث ان العمل المصنفات تقوم عليها،هذا من جهة ومن جهة اخرى ان جل محركات البحث خورزميات البحث الحديثة تعتمد عليها تستخدم هذه الطريقة الان اسرع واقوى اما الجميل فيها هي ان تعتمد على مبداء " اكتب القليل وافعل الكثير " .

اين توجد ؟؟؟

في ما له علاقة بالتحقق و الامن الالكتروني مثلا :

تأكد من نحو البريد الالكتروني صحيح بطاقات الاعتماد وحتى سيريال ... .

تحقق من حقول الادخال خالية من الرموز المشبوهة "+,-,/,*,|,?,{},<>....." وغيرها

ولتعرف المزيد جرب هذا :

http://www.gskinner.com/RegExr

http://www.regular-expressions.info

الاساسيات :

توجد وثقية رائع للاخ **محمد محمود فؤاد** http://www.eng-mmf.com

كيفية الاستعمال في السي :

توجد مكتبة في السي تحت اسم regex.h يتم استدعائها وبرنامج يكون كالتالي :

```
#include <regex.h>
#include<stdio.h>

int main (int argc, char *argv[])
{
regex_t rg1;
char *pat1="1",*str1="2";
regcomp(&rg1,pat1,3);
if (regexec(&rg1,str1,0,NULL,0)==0) {
        printf("%s Matches %s\n",str1,pat1);
}
 else {
        printf("No Match\n");
}
regfree(&rg1);

       return 0;
}
```

A

B

C

D

الشرح :

1: التعبير القياسي

2: النص المراد تطبيق عليه

3:وسيط للدالة ويكون من الجدول التالي كما يمن تمرير اكثر من وسيط عن طريق اضافة " | "

REG_EXTENDED	قبول التعابير الإضافية
REG_ICASE	إهمال حالة الحروف (الإنجليزية) إن كبيرة أو صغيرة
REG_NOSUB	عدم توليد السلاسل الفرعية(تلك المحصورة بين أقواس هلالية)
REG_NEWLINE	كحالة خاصة "\n"معاملة السطر الجديد

A : تركيب يتم فيه التعابير والنصوص القياسية

B : دالة يتم تمرير لها التعبيرالقياسي مع عنوان تركيب الذي يتم تخزين فيه التعابير والنصوص القياسية والوسيط الثالث تم شرحه سابقا

C : دالة تقوم بتنفيذ التعابير والنصوص القياسية على النص المدخل او المراد وسنرى امثلة للتوضيح اكثر
D : لتحرير الذاكرة
الامثلة :

```
#include <regex.h>
#include<stdio.h>

int main (int argc, char *argv[])
{
regex_t rg1;

char *pat1="epidemic",*str="EpIdEmIc";
regcomp(&rg1,pat1,REG_ICASE);
if (regexec(&rg1,str1,0,NULL,0)==0) {
        printf("%s Matches %s\n",pat1,str1);
}
 else {
        printf("%s No Match\n %s",pat1,str1);
}
regfree(&rg1);

       return 0;
}
```

```
[aye7@localhost ~]$ ./reg
epidemic  Matches EpIdEmIc
```

```
#include <regex.h>
#include<stdio.h>

int main (int argc, char *argv[])
{
regex_t rg1;

char *pat1="epidemic",*str="Aye7";
regcomp(&rg1,pat1,REG_ICASE);
if (regexec(&rg1,str1,0,NULL,0)==0) {
        printf("%s Matches %s\n",pat1,str1);
}
 else {
        printf("%s No Match\n %s",pat1,str1);
}
regfree(&rg1);

       return 0;
}
```

```
[aye7@localhost ~]$ ./reg
epidemic  No Match Aye7
```

```
#include <regex.h>
#include<stdio.h>

int main (int argc, char *argv[])
{
regex_t rg1;

char *pat1="P",*str="epidemic";
regcomp(&rg1,pat1,REG_ICASE);
if (regexec(&rg1,str1,0,NULL,0)==0) {
        printf("%s Matches %s\n",pat1,str1);
}
 else {
        printf("%s No Match\n %s",pat1,str1);
}
regfree(&rg1);

       return 0;
}
```

```
[aye7@localhost ~]$ ./reg
P Matches epidemic
```

```
#include <regex.h>
#include<stdio.h>

int main (int argc, char *argv[])
{
regex_t rg1;

char *pat1="^P",*str="epidemic";
regcomp(&rg1,pat1,REG_ICASE);
if (regexec(&rg1,str1,0,NULL,0)==0) {
        printf("%s Matches %s\n",pat1,str1);
}
 else {
        printf("%s No Match\n %s",pat1,str1);
}
regfree(&rg1);

       return 0;
}
```

```
[aye7@localhost ~]$ ./reg
P  No Match epidemic
```

الفصل السادس:

العمليات أو المعالجات PROCESSEC

الفصل الثامن : الخيوط وتعدد المسالك MultiThreading