بسم الله الرحمن الرحيم

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '    %s [label="%s' % (nodename, label),
    if isinstance(ast[1], str):
        print ', %s' % ast[1]
        else:
            print '"];'
    else:
        print '"];'
        children = []
        for  in n, childenumerate(ast[1:]):
            children.append(dotwrite(child))
        print ,'    %s -> {' % nodename
        for  in :namechildren
            print '%s' % name,
```

CSE131: Computer Programming

Chapter (4)

﴿وَقل رَّبِّ زِدْنِي عِلْماً﴾

(طه: ١١٤)

* Functions :

بلجأ لل functions لو عايز اكتب Code
بيعمل عملية معينة كذا مرة، فبعمل
function تعمل العملية دي بدل تكرار
كتابة ال Code بتاع العملية

تنبيه : ميتفعش اعرف function جوه function
ده يعتبر syntax error

① Without return function

دي عبارة عن function بتعمل
عملية معينة ومش بترجعلي قيمة

```
void myfunction()
{
...
}
```

- نوع ال function ← void
- اسم ال function ← myfunction
- ... → function body : بكتب فيه العملية اللي عايز انفذها

② With return function

دي عبارة عن function بتعمل
عملية معينة وبترجعلي قيمة
محتاجاها في البرنامج ، فال function
بتروح على جنب تحسب القيمة دي
وتيجي تدهالي عن طريق ال return

```
int myfunction(int x)
{
...
return m;
}
```

- [int myfunction(int x)] → function prototype : طريقة تعريف ال function
- input parameter : القيمة ال هدخلها في ال main وتخش ال function تنفذ العملية
- return value : دي القيمة ال هترجعلي من العملية ال في ال body

* Function Call

الطريقة ال بنادي بيها على ال function جوه ال main

```
myfunction(5);
```

القيمة ال هتخش ال function

* Function Prototype

① ممكن اعرفها بالتفصيل قبل ال main

```
void myfunction()
{
...
}
void main()
{
...
}
```

② ممكن اعرفها بالتفصيل بعد ال main
بشرط أحط لوجودها قبل ال main

```
void myfunction();
void main()
{
...
}
void myfunction()
{
...
}
```

متنساش دي ←

* Variable Scope :

① local Variable :

← ده عبارة عن Variable متعرف جوه Function معينة ،
اول ما تخرج منها مش بيبقى له وجود "بيموت"

```
void myfunction()
{
int x = 5;
}
void main()
{
int x = 3;
}
```

- هنا ال x=5
- هنا ال x=3
- تخرج من ال function قيمة ال x تموت

② Global Variable :

← ده عبارة عن Variable متعرف خارج اي Function
و قيمة معروفة في اي حتة

```
int g_x = 5;
void main()
{
printf("%d",x);
}
void myfunction()
{
printf("%d",x);
}
```

- هيطبع 5
- يفضل اكتبها كدا عشان السمبول يبقى واضح انه global

③ Static Variable :

← ده معناه أن ال Variable بيحتفظ بالتغيير ال
حصل فيه

```
void main()
{
myfunction()
}
void myfunction()
{
int x = 0;
static int s_y = 0;
printf("%d %d",x,s_y);
x++;
y++;
}
```

- يفضل اكتبها كدا عشان السمبول يبقى واضح انه static
- * هنا الزيادة هتتنسى اول ما أخرج من ال function ولو دخلتها تاني هتم تعريف ال x بالقيمة الابتدائية المكتوبة
[ده بالنسبة ال automatic]
- دي مختلف : عملية ال initialization بتتم مرة واحدة وال y بتحتفظ بكل قيمة جديدة تاخدها حتى لو خرجت

* Recursion : مفهوم معناه أن ال function بتنادي نفسها

direct recursion

← ال function بتنادي نفسها جوه نفسها

```
void function1()
{
...
function1();
}
```

indirect recursion

← function بتنادي function تانية جواها ال function الاولانية

```
void function1()
{
...
function2();
}
void function2()
{
...
function1();
}
```

① infinite loop printing

*حاجة ملهاش قيمة

Ⓐ normal loop

← معناه أن أعمل infinite loop بنفس أنه ينفذ عملية معينة

Ⓑ Recursive loop

← معناه أن ال function تفضل تنادي نفسها من غير ما حاجة توقفها

② finite loop printing

*مهم

Ⓐ normal loop

← معناه أن أعمل loop بنفس أنه ينفذ عملية معينة

Ⓑ Recursive loop

← معناه أن أحدد عدد مرات معين لل function أنها تنادي نفسها عن طريق الجمل الشرطية مثلاً

* Macro :

بعمل replace لل code او constant
علشان أسرع الكتابة

المميز أن ال Macro مش بيفرق معاها ال data type

مثل :

⓪ object-like macro

```
#define PI 3.14
```

بدل ما كل شوية اعرف ال π
بعرفها مرة واحدة بالطريقة دي

② function-like macro

```
#define ADD(a,b) (p) + (q)
```

- اسم ال Function ← ADD
- Parameters ← (a,b)
- العملية ← (p) + (q)

```
void main()
{
printf("%d", ADD(3,5));
}
```

Run ⇒