Week10

Call back : 可以控制function按照使用者要求的先後順序作執行(不考)

def A(): def A(B):

print(“A”) B()

def B(): print(“A”)

print(“B”) def B():

print(“B”)

印出

A B  
B A

Context free grammar : 與上下文無關的文法

與上下文有關的意思是 需要先做完上面的判斷 產生的結果才能去做下面的判斷

Ex暫存器存的變數值有相依性

BNF vs. EBNF:

BNF <E> ::= <T> | <E> + <T> | <E> - <T>

EBNF <E> ::= <T> { ‘+’ <T> | ‘-‘ <T> }

存在著left recursion才會用EBNF

Terminal(token) vs. Non-terminal

隨便拿一個non-terminal，可以轉變成terminal or non-terminal的組合

<term> ::= <factor> or <term> ::= <term> ‘\*’ <factor>

Leftmost(rightmost) defivation : 左結合右結合的derive方式 (?不確定)

If X = 0 then if Y = 0 then S1 else S2 這個在PASCAL是找相近的

If X == 0:

If Y == 0:

S1()

Else:

S2()

A=B=C C assign 給B會得出一個值(C本身的值)，再把這個值assign給A

一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

(投影片只有ID 跟 NUM)

(幹這好像不會考)

Lexeme vs. token:

Lexeme : 所輸入的字串 數字 浮點數都是

Token : 在做完regular expression後的產物

。lexical analyzer // scanner

。Regular expression <- 進來的時候叫做Lexeme

。Extend Regular expression ->出來的時候叫做token

。state diagram // 系統轉換的圖 用圖的方式表達所做的操作流程

。finite automata //呈現系統當下的狀態及接下來要做的事(ATM顯示目前金額)

。Syntactical analyzer //parser

。Recursive decent parsing

1. 不能有left recursion
2. 一個展開式的各個case的 “the set of first tokens”沒有交集

六元項屬性

1. Name:由使用者所命名之identifier
2. address:

。同一個變數再不同的地方或不同的時間可能會對應到不同的變數，像是loading到memory的時候每次都在不同位置

。別名(aliases):當不同的變數名稱引用同一個memory address

(pointer指向同一塊空間，像是array的index)

當2個ptr指向同一塊空間，如果其中一個對此空間作操作，另外一個幾乎不會察覺，可能會因此error

1. Type:決定變數值的範圍以及可作用在該變數之運算
2. Value:變數所關聯的記憶體內容 // int 二進位 , float IEEE754 解讀 儲存不同
3. Lifetime:變數所擁有記憶體的這一段時間

區域變數:在runtime被呼叫的時候

Static變數:在compile時就被定義好了

1. Scope:在城市中可以存取某一變數的有效範圍

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

Week11

Binding:一個程式實體(Program entity)與其屬性發生關聯(association)稱之

Binding可能發生的時期

設計: binding可能的型別 ex : x = x + 10 | x = x + 10.2

實作: binding 用何種硬體實作

編譯: binding型別 by Programmer or Compiler

載入: binding specific memory location ，obj檔載入到記憶體

連結: binding memory address for external variables

Call function: local definition, pass parameters…….

Any time: binding any value

Sub{ int a; } or Sub(int x , int a) { int a = 10; }

這個sub function被執行時 a才會產生

Static binding vs. dynamic binding

Static binding是name binding發生在程式執行之前(compile time)

Dynamic binding發生在程式正在執行的時候(run time)

如果compiler允許動態調整型別 那就不能依賴compiler偵錯 就需要常常得對此變數做檢查 看甚麼時候做了變動

Binding 是將一個名稱與某個值、物件或函式關聯起來的過程

(2個解釋都看一下)

可以被分為 Static binding（在編譯時期完成）或 dynamic binding（在運行時期進行）。

Static binding 通常發生在靜態語言中，如C或Java。

dynamic binding 則通常發生在動態語言中，如Python或JavaScript。允許在執行時期根據程式的上下文和運行時狀態來決定具體的綁定關係。

Dynamic type binding

依照當下執行的指令來決定變數是甚麼型別

Interpreter較優

Static type binding

Explict宣告 大部分都是外顯式 中間如果要更改型別是不被允許的

缺點 在使用前就要把東西都宣告好較麻煩

Implicit 宣告 變數使用錢的data type 、 address 、 值都需要先被綁定

如果有變數名稱單詞拼錯 會被視為是新的變數名稱 很難找出錯誤

缺點

雖然不用事先給型別 會根據系統規則給型別 但有可能被給的型別不是所需要的

A = 5 -> int

A = A + 3.14 -> A變成float 可能會有精度的問題

// 要增加程式可讀性不能只使用自己熟悉的語言用法

// 非必要的話不要把程式碼複雜化 越簡潔越好

Week 12

Is it possible for a variable

。Not to have a name

有可能，存放在heap的memory address，透過ptr去指向這一塊空間，他是一個變數，但是他沒有名字

。Not to have an address? ( = does not exist )

有可能，compiler在compile階段知道有這個變數的存在 但是沒有給予地址，只是預留空間

。To have different addresses at different times?

有可能

1. 對function call 來說，function call的變數在不同的時間點被呼叫，會有不同的地址
2. 像python有強制轉型，不同型別儲存值所需的長度不同，可能會影響到address
3. 每一次load程式到memory的時候會因為起始位置的不同，而導致每次的address都不同

。Not to have a value

不太可能，就算不在宣告的時候賦值，編譯器會自動賦值，但初始值會不知道是多少

Storage binding

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

Static variable

History-sensitive variables = local variables 會保留上一次的狀態 不會因為function結束而被重製

優點 vs. 缺點

static變數在compile階段就被放在data segment中是優點也是缺點

static變數不支援recursion 因為recursion是要根據傳入的參數做操作

但是static變數會保留之前的狀態 怎麼改都一樣

Stack-dynamic variable

也就是區域變數，當function進到call stack，variable也進去stack，當funtion不被呼叫，variable也就不會進去占用stack

好處 vs. 壞處

不占用memory資源 不像static 變數會提供上一次的結果

支援recursion

Explicit heap-dynamic variable // new

優點 vs. 缺點

可以手動決定何時要用這塊空間 如果忘記釋放空間，沒有被指向的記憶體空間就會一直留在memory

Implicit heap-dynamic variable // =

優點 vs. 缺點

不用這塊空見會自動做記憶體的釋放 實作成本高 需多花時間檢查空間背後的變數 不能手動決定能不能使用這塊空間

//上述優缺點比較可能會考 給一段程式碼 問說是屬於哪幾類 有甚麼優缺

Static scope 在編譯時就已經確定一個變數的 scope，不會隨著程式的執行而改

變。這種解析方式在大多數主流程式語言中被採用，例如C++、Java、Python

等。

當全域和區域變數名稱發生衝突，那在function內做運算時會先以function

為

主

Dynamic scope 根據function被呼叫的次序，決定變數的scope，這種的解析方

式在 Lisp 中被採用。

當全域和區域變數名稱發生衝突

C / C++ :

Int x = 20;

Void func() {

Int x = 10; // local

Cout << x; // visible 印出10, 而global x 變成hidden

Cout << ::x ; // global 印出20

}

Java

class Parent {

int num = 100;

}

class Child extends Parent {

int num = 200;

void printNum() {

System.out.println("Child num: " + num);

System.out.println("Parent num: " + super.num);

}

}

C / C++

.h : declaration 僅宣告，不配置記憶體，可以多個.h檔

包成.h檔，廣為他人所使用，可以看成工具列表

Printf: 當compiler掃到未被宣告的function 或指令就會去看有宣告的.h檔中printf這道指令有沒有被定義

.c and .cpp : definition 只能有一個.c | .cpp

extern int x;

是整數變數x的 Declaration，它通知編譯器該變數存在，但不提供其具體的定義。

int x = 5;

是整數變數x的 Definition，它確定了該變數的名稱、類型和初始值。

但是可能會存在extern 跟 static的衝突

A: cpp 如果有一個static int x ，那就算B、C宣告了extern x 也不能夠使用到x

B: cpp 會被當成undefined symbol

C: cpp

Week 13

#define X 10 vs. const int X = 10;

前者C preprocessor在編譯前會把整隻程式有X的地方都用後面的10取代

後者會有型別上的安全 定義的時候需要選擇型別

Function的溝通希望是用local變數少用全域變數，避免和他人程式上有名稱的衝突，希望可以從變數名稱看出他是歸屬在哪 :

Global : int gtemp;

Data member: int mtemp;

Coercion（強制轉換）是指compiler在程式執行期間自動轉換一個資料型別為另一個資料型別，以符合某種操作或運算的需求。

像是有一個function(int x) ，但是傳入了float的參數，轉成int但可能會有warning，結果可能會不如預期

Coercion 通常是 Implicit 的，不需要明確的程式碼來指示轉換。

Cast : explicit type conversion

Float a = 0.2;

Int b = (int) a;

Primitive data type:像是 int , char, Double, Boolean可以直接取值的 int a = 5;

但pointer是間接取值就不是primitive

a = b vs. if (a ==b)

前者就assign

後者就是看是不是同個primitive

不同primitive : 看a所指向的memory address 和b 的address是否相同

同primitive : 看a所指向的memory address 和b的address內容是否相同

Enum

在co-work的時候，enum的好處就是命名的常數name，只要enum後的名稱不一樣就不會有衝突

enum A{ enum B {

test1 = 0; test1 = 0;

test2 = 1; test2 = 1;

} }

型別上的安全問題，define不檢查型別，enum具有型別上的安全，限制在enum{}的範圍嫩，只能去做比較TokenType裡的常數值

Debug可控性、Scope

Associative array

Python:

Inventory = {“noodles”:3, “pork”:2, “beef”:2}

Inventory[“rice”] = 5

Static array : 在compile階段就配置好空間跟memory address

Fixed stack-dynamic array : 固定大小的function中的array，和function同進同出

Stack-dynamic array:大小未被是先設定，沒辦法事先知道array有多大

Ex:

Void func( int n ) {

Int a[n];

}

Fixed heap-dynamic array: 像是使用new()、malloc()產生固定大小的array //delete 、 free

heap-dynamic array:根據傳進的參數值決定array的大小

jagged array: [[ 1,2,3] , 4, [5] ] 一般來說對array每個index大小都會是相同的，但是python的list不一樣，是可以有不同大小的index

Array是指標常數 不支援像ptr那樣做記憶體位置的變動

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

自動產生的描述

Week 14

Q:Array的配置是何時發生?

A:大部分是在runtime 才配置，runtime時可能只是檢查array大小，不配置

Q : A[5:4]，第一個元素為A[0,0]， I0 = 1000，d = 1，求A[3,2]?

A: 可看成A[0:5, 0:4]

Row [(3-0)\*(4-0+1) + (2-0) ]\* 1 + 1000

Column [(2-0)\*(5-0+1) + (3-0) ]\* 1 + 1000

三維陣列

B[ 1:x , 1:y, 1:z ] , address of B[s, t, u]

Row [(s-1)\*y\*z + (t-1)\*z + (u-1) ]\* sizeof(Type) + Base address

Column [(u-1)\*x\*y + (t-1)\*x + (s-1) ]\* sizeof(Type) + Base address

Union 不考 跳過(^^)

Dangling pointer 是空間被Release了卻還是指向那塊空間的指標

假設有int \*p1, \*p2,\*p3去指向同一塊空間，當p1把這塊空間release了，卻沒有去對p2,p3做相對應的動作

應對方法

Tombstones : 透過Tombstone去指向這塊空間，由tombstone去access這塊空間，remove後會自動指向NULL

Locks-and-keys =>(key, address) 在指向空間時會生成一個key值，能夠去比對，但是很耗時，C裡面是不做這件事，成本會提高

Week 15

Lost heap-dynamic variables(new出、malloc出的變數) :

Garbage collection 、memory leak等等的問題

Heap management(可以看作是上述的解決方法)

Mark sweep : 當空間分配出去 如果被Pointer指向，就作mark；反之則無

從根物件開始，遞迴地訪問和標記所有與根物件直接或間接相連的物件。

未被標記的物件將會被 garbege collection 回收。

優點:

能夠處理循環引用的情況，即使物件之間存在相互引用，只要這些物件不再被根物件引用，就可以被回收。

缺點:

Mark sweep 需要將整個程式暫停，導致較長的停頓時間。

Reference count :每隔一段時間作garbage collection，就去更新reference count，如果多一個ptr指，ref count就加一。

只要有一個pointer指向這塊空間，那這塊空間就不能回收

優點:

能夠即時回收不再使用的物件，不需要暫停程式。

缺點:

需要維護每個物件的參考計數，並且需要額外的開銷來更新和維護計數器。

Type checking

1. Static Type Checking

發生在編譯時期，程式碼編譯前檢查變數、運算符和表達式的類型是否符合程式語言的型別規範。

2. Dynamic Type Checking

發生在程式碼執行時期，當程式碼執行到特定的操作時，會檢查變數、運算符和表達式的類型是否符合預期的類型。一張含有 文字, 螢幕擷取畫面, 軟體, 字型 的圖片

自動產生的描述

不使用間接運算子”\*” // implicity deferenced

Ref 和 count 指向同一個位址 ….

=== vs. ==

!== vs. != 除了比較內容之外也會比較型別是否不同

Pointer vs. reference

Pointer可以在memory之間游走

Reference不使用”\*”的情況下就可以去存取object的東西

Referential Transparency : 將兩個expression順序對調都不會影響結果

Short-circuited evaluation :

在計算的過程當中，只要一個計算的結果會影響整個判斷式的結果，就具備這個性質，後面的式子就不需要去判斷了，在coding時要注意語言是否具備此特性ex: C、 JAVA

一張含有 文字, 螢幕擷取畫面, 功能表, 白板 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述