Compiler Designs

Programming HW2: parsing and C code generation

四電子四丙 B10902227 吳宗喬

1. 使用方法

(一). 進入[Compiler2024]B10902227-codegenerate 資料夾，
輸入 make 會編譯並產生執行檔 codeGenerator

(二). 執行 codeGenerator (輸入格式: ./codeGenerator **sample.qv**)
(**sample.qv** 為自定的 qv 檔)

(三). 會產生 gen.c 的程式碼，即本次作業的輸出

2. 實現部分

本程式中我實現 real/int 的資料型態，並支援+-*/=等運算，支援函數，無限維度的陣列，陣列的四則運算，變數型態確認，變數作用域，基礎 for-loop(因為不支援 bool 型態所以雖然可以寫，但很難用)，而尚未實現的部分為函數輸入的變數型態確認與函數回傳值的確認，在 Expression 的表示上是完整的。

Expression 轉換是根據 C 優先級，但為保證 qv 的執行順序跟 C 執行的順序相同，Expression 都會以中括號括住以保證優先順序(EX: a+b*3 →(a+(b*3)))。

矩陣的運算在轉成 C 後會變成一堆用 index 處理的單元，而不是以 for-loop 的方式轉換(EX: a={1,2}; b={3,4}; a+b → a[0]+b[0], a[1]+b[1])。

所有的常數如果可以計算化簡的部分在轉成 C 後都會是化簡過的數字，而若 Expression 包含變數無法化解的則會保留(EX: a+3*4+c → ((a+12)+c))。

函數可以用矩陣變數或常數進行初始化，但不可以用矩陣常數初始化(因為 C 也不准)，且函數返回值不可為矩陣(EX: a={1,2}; f(a)可以 f({1,2})不可以) 。

生成的 C 標頭檔自帶 stdio.h 與 stdlib.h。

轉換會根據 scoping 以 tab 進行排版。

Print 與 println 為初始就可使用的函數，只能單一輸入，但可以輸入變數，字串，矩陣變數。

實作上除了多維矩陣運算外都遵守 C 的限制。

3. 例子(左圖為 qv 右圖為 C)

以下先以老師提供的例子測試:

(1).

```
fun main () {
    var i: int;
    i = 10;
    print(i);
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main (){
    int i;
    (i=10);
    printf("%d",i);

}
```

如右上，i=10 會轉換成(i=10)，雖然很醜但可以保證正確。

(2).

```
fun main () {
    var i: int = 10;
    var j: real = 3.1415;
    print(i);
    print(j);
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main (){
    int i = 10;
    float j = 3.1415;
    printf("%d",i);
    printf("%f",j);

}
```

print 函數會根據宣告選擇%d，%f 或%s

(3).

```
fun main () {
    var i: int = 10;
    var j: real = 3.1415;
    var k: int = i + j;
    var l: real = i + j;
    print(k);
    print(l);
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main (){
    int i = 10;
    float j = 3.1415;
    int k = (i+j);
    float l = (i+j);
    printf("%d",k);
    printf("%f",l);

}
```

變數具有 real/int 等型別

(4).

```
fun main () {
    var radius: real = 5;
    var pi: real = 3.1415;
    var area: real = radius * radius * pi;
    print(area);
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main (){
    float radius = 5;
    float pi = 3.1415;
    float area = ((radius*radius)*pi);
    printf("%f",area);

}
```

同 C 的四則運算系統

(5).

```
fun main () {
    var i: real = 1.5;
    var j: real = 3.14;
    var k: real = 2.8;
    print(i + j * k);
    print("\n");
    print(i * (j + k));
    print("\n");
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main (){
    float i = 1.5;
    float j = 3.14;
    float k = 2.8;
    printf("%f",(i+(j*k)));
    printf("%s","\n");
    printf("%f",(i*(j+k)));
    printf("%s","\n");

}
```

(6).

```
fun main () {
    print(123.456);
    println(123.456);
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main (){
    printf("%f",123.456);
    printf("%f\n",123.456);

}
```

支援換行 print

(7).

```
fun main () {
    var vi1: real[5] = {5, 3, 4, 1, 2}; // vi1[0]~vi1[4]
    var vi2: real[5] = {2, -2, 4}; // Missing dimensions assumed 0s
    print( vi1 * vi2 ); // Inner product, output: "20"
    print( "\n" );
    print( vi1 + vi2 ); // Dimension-wise addition, output: "{7, 1, 8, 1, 2}"
    print( "\n" );
}
```

```
./codeGenerator.out sample.qv
ERROR: assign different dimensions
```

維度不同的矩陣無法編譯

(8).

```
fun main () {
    var vi1: real[5] = {5, 3, 4, 1, 2}; // vi1[0]~vi1[4]
    var vi2: real[3] = {2, -2, 4}; // Missing dimensions assumed 0s
    print( vi1 * vi2 ); // yyerror("ERROR: mismatched dimensions")
    print( "\n" );
}
```

```
./codeGenerator.out sample.qv
ERROR: mismatched dimensions
```

維度不同的矩陣無法賦值

(9).

```
fun main () {
    var arr1: real[2][2] = {1, 2, 3, 4}; // arr1[0][0], arr1[0][1], arr1[1][0], arr1[1][1]
    var arr2: real[2][2] = {1, 0, 0, 1}; // arr2[0][0], arr2[0][1], arr2[1][0], arr2[1][1]
    print( arr1 * arr2 ); // {1, 2, 3, 4}
    print( "\n" );
    print( arr1 + arr2 ); // {2, 2, 3, 5}
  print( "\n" );
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main (){
    float arr1[2][2] = {1 , 2 , 3 , 4};
    float arr2[2][2] = {1 , 0 , 0 , 1};
    printf("%f, %f, %f, %f",(arr1[0][0]*arr2[0][0]), (arr1[0][1]*arr2[0][1]), (arr1[1][0]*arr2[1][0]), (arr1[1][1]*arr2[1][1]));
    printf("%s","\n");
    printf("%f, %f, %f, %f",(arr1[0][0]+arr2[0][0]), (arr1[0][1]+arr2[0][1]), (arr1[1][0]+arr2[1][0]), (arr1[1][1]+arr2[1][1]));
    printf("%s","\n");

}
```

將 qv 的矩陣運轉成 C 後變成元素個別運算

(10).

```
fun main () {
    var arr1: real[2][2] = {{1, 2}, {3, 4}}; // arr1[0][0], arr1[0][1], arr1[1][0], arr1[1][1]
    var arr2: real[2][2] = {{1, 0}, {0, 1}}; // arr2[0][0], arr2[0][1], arr2[1][0], arr2[1][1]
    print( arr1 * arr2 ); // {1, 2, 3, 4}
    print( "\n" );
    print( arr1 + arr2 ); // {2, 2, 3, 5}
  print( "\n" );
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main (){
    float arr1[2][2] = {1 , 2 , 3 , 4};
    float arr2[2][2] = {1 , 0 , 0 , 1};
    printf("%f, %f, %f, %f",(arr1[0][0]*arr2[0][0]), (arr1[0][1]*arr2[0][1]), (arr1[1][0]*arr2[1][0]), (arr1[1][1]*arr2[1][1]));
    printf("%s","\n");
    printf("%f, %f, %f, %f",(arr1[0][0]+arr2[0][0]), (arr1[0][1]+arr2[0][1]), (arr1[1][0]+arr2[1][0]), (arr1[1][1]+arr2[1][1]));
    printf("%s","\n");

}
```

二為矩陣運算

(11).

```
fun main () {
    var i: int;
    var i: real[3] = {2, -2}; // yyerror("ERROR: duplicate declaration")
}
```

```
Error: variable "i" duplicate declaration
```

同 scoping 中重複宣告會無法編譯

(12).

```
fun main () {
    var vi: real[2] = {2, -2, 5}; // yyerror("ERROR: too many dimensions")
}
```

```
ERROR: assign different dimensions
```

維度不同無法初始化

(13).

```
fun main () {
    var i: real = 3.0;
    {

        var i: int = 2;
    }
    var i: real[2]; // yyerror("ERROR: duplicate declaration");
}
```

```
Error: variable "i" duplicate declaration
```

在同個 scoping 內重複宣告

(14).

```
fun add(i: int, j: int): int { // Add i and k, then return the result.
  ret i + j;
}

fun main() { // For simplicity, main() always returns 0
  print(add(3, 5));
  print("\n");
}
```

```
#include<stdio.h>
#include<stdlib.h>
int add (int i, int j){
    return (i+j);

}
void main (){
    printf("%d",add(3,5));
    printf("%s","\n");

}
```

可以使用需告過的函數

(15).

```
fun add(i: int, j: int): int { // Add i and k, then return the result.
  ret i + j;
}

fun main() { // For simplicity, main() always returns 0
    for(var i:int=10; i; i=i-1){
        for(var j:int=10; j; j=j-1){
            print(add(i,j));
        }
    }
}
```

```
#include<stdio.h>
#include<stdlib.h>
int add (int i, int j){
    return (i+j);

}
void main (){
    for(int i = 10;i;(i=(i-1))){
        for(int j = 10;j;(j=(j-1))){
            printf("%d",add(i,j));

        }

    }

}
```

支援 for-loop 但無法使用 bool

(16).

```
fun main() { // For simplicity, main() always returns 0
    var A : real = 1.0;
    var B : int[3] = {1,2,3} ;
    B[A] = 3 ;
}
```

```
Error: invalid index(float).
```

浮點數不可做 index

(17).

```
fun main() { // For simplicity, main() always returns 0
    var A : real[2][2][2][2] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16} ;
    A = A + A + A;
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main (){
    float A[2][2][2][2] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , 15 , 16};
    A[0][0][0][0] = ((A[0][0][0][0]+A[0][0][0][0])+A[0][0][0][0]),A[0][0][0][1] = ((A[0][0][0][1]+A[0][0][0][1])+A[0][0][0][1]),A[0][0][1][0] = ((A[0][0][1][0]+A[0][0][
}
```

任意維度矩陣運算

(18).

```
fun main(argc:int, argv:int[10]): int {
    argv[0] = 123;
    ret argc ;
}
```

```
#include<stdio.h>
#include<stdlib.h>
int main (int argc, int argv[10]){
    (argv[0]=123);
    return argc;

}
```

支援帶變數的函數與回傳值

(19).

```
fun main(argc:int, argv:int[10]): int[10] {
    argv[0] = 123;
    ret argc ;
}
```

```
Error: unsupport multiple dimensions return.
```

不支援回傳矩陣值

(20).

```
fun main() {
    var A : real[2][2] = {1,2,3,4} ;
    var B : real[2][2] = {{1,2},{3,4}} ;

}
```

```
#include<stdio.h>
#include<stdlib.h>
void main (){
    float A[2][2] = {1 , 2 , 3 , 4};
    float B[2][2] = {1 , 2 , 3 , 4};

}
```

兩種矩陣初始化的方式都可接受(因為 C 也可以)

(21).

```
fun main() {
    var A : real[2][2] = {1,2,3,4} ;
    var B : real[2][2] = {{1,2},{3,4}} ;
    A + B + {1,2,3,4};
}
```

```
ERROR: mismatched dimensions
```

但在非初始化的地方無法這樣方便使用

(22).

```
fun main() {
    var A : real[2][2] = {1,2,3,4} ;
    A = A + {{5,6},{7,8}};
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main (){
    float A[2][2] = {1 , 2 , 3 , 4};
    A[0][0] = (A[0][0]+5),A[0][1] = (A[0][1]+6),A[1][0] = (A[1][0]+7),A[1][1] = (A[1][1]+8);

}
```

支援右值建立的矩陣進行運算

(23).

```
fun A(a : int) :int{
    ret 1 ;
}
fun B(a : int):int{
    ret 1 ;
}
fun C(a : int):int{
    ret 1 ;
}
fun D(a : int):int{
    ret 1 ;
}
fun main() : int {
    var E : real[2][2] = {1,2,3,4} ;
    E[D(C(B(A(1))))][1+2*3/4+5-6*7/8];
    ret 0;
}
```

```
#include<stdio.h>
#include<stdlib.h>
int A (int a){
    return 1;

}
int B (int a){
    return 1;

}
int C (int a){
    return 1;

}
int D (int a){
    return 1;

}
int main (){
    float E[2][2] = {1 , 2 , 3 , 4};
    E[D(C(B(A(1))))][2];
    return 0;

}
```

複雜的括號結構與四則運算

同 C，若運算中有 float 才會隱式轉成 float

(24).

```
fun main() : int {
    var E : real[2][2] = {1,2,3,4} ;
    print(E) ;
    println(E) ;
    ret 0;
}
```

```
#include<stdio.h>
#include<stdlib.h>
int main (){
    float E[2][2] = {1 , 2 , 3 , 4};
    printf("%f, %f, %f, %f",E[0][0], E[0][1], E[1][0], E[1][1]);
    printf("%f, %f, %f, %f\n",E[0][0], E[0][1], E[1][0], E[1][1]);
    return 0;

}
```

矩陣的 print 與 println

(25).

```
fun main() : int {
    var E : real[2][2] = {1,2,3,4} ;
    add(E) ;
    ret 0;
}
```

```
Error: Undeclared variable "add".
```

無法使用未宣告的函數

(26).

```
fun main() : int {
    var A : real[3] = {1,2,3} ;
    A=({1,2,3}+{2,3,4})*A+{8,4,3}/{1,2,3}*({2,5,2}+A);
}
```

```
#include<stdio.h>
#include<stdlib.h>
int main (){
    float A[3] = {1 , 2 , 3};
    A[0] = ((3*A[0])+(8*(2+A[0]))),A[1] = ((5*A[1])+(2*(5+A[1]))),A[2] = ((7*A[2])+(1*(2+A[2]))));

}
```

複雜矩陣運算

(27).

```
fun main() : int {
    var A : int[3] = {1,2,3} / {0,1,1};
}
```

```
ERROR: zero divisor (int)
```

防止常數除零

(28).

```
fun main() : int {
    val A : int ;
    var B : int ;
}
```

```
#include<stdio.h>
#include<stdlib.h>
int main (){
    const int A;
    int B;

}
```

val 與 var 分別轉成一般與 const 變數

(29).

```
fun main() : int {
    var A : int ;
    {
        var A : real ;
        A = 1.0 ;

    }
    print(A) ;

}
```

```
#include<stdio.h>
#include<stdlib.h>
int main (){
    int A;
    {
        float A;
        (A=1.0);

    }
printf("%d",A);

}
```

變數遵守 scoping 的範圍

(30).

```
fun printSum(a : real[10]) {
    var c : int [10] = a / a ;
    println(c) ;
}
fun main() : int {
    var A : real [10] = {0,1,2,3,4,5,6,7,8,9} ;
    var B : real [10] = A*A ;
    var C : real [10] = A+B*A ;

    println(A) ;
    for(var i: int=10; i; i=i-1){
        A[i] = A[i] + i ;
    }
    println(A) ;
    println(A * C) ;

    printSum(A) ;

}
```

```
#include<stdio.h>
#include<stdlib.h>
void printSum (float a[10]){
    int c[10] = {(a[0]/a[0]) , (a[1]/a[1]) , (a[2]/a[2]) , (a[3]/a[3]) , (a[4]/a[4]) , (a[5]/a[5]) , (a[6]/a[6]) , (a[7]/a[7]) , (a[8]/a[8]) , (a[9]/a[9])};
    printf("%d, %d, %d, %d, %d, %d, %d, %d, %d, %d\n",c[0], c[1], c[2], c[3], c[4], c[5], c[6], c[7], c[8], c[9]);

}
int main (){
    float A[10] = {0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9};
    float B[10] = {(A[0]*A[0]) , (A[1]*A[1]) , (A[2]*A[2]) , (A[3]*A[3]) , (A[4]*A[4]) , (A[5]*A[5]) , (A[6]*A[6]) , (A[7]*A[7]) , (A[8]*A[8]) , (A[9]*A[9])};
    float C[10] = {(A[0]+(B[0]*A[0])) , (A[1]+(B[1]*A[1])) , (A[2]+(B[2]*A[2])) , (A[3]+(B[3]*A[3])) , (A[4]+(B[4]*A[4])) , (A[5]+(B[5]*A[5])) , (A[6]+(B[6]*A[6])) , (A
    printf("%f, %f, %f, %f, %f, %f, %f, %f, %f, %f\n",A[0], A[1], A[2], A[3], A[4], A[5], A[6], A[7], A[8], A[9]);
    for(int i = 10;i;(i=(i-1))){
        (A[i]=(A[i]+i));

    }
printf("%f, %f, %f, %f, %f, %f, %f, %f, %f, %f\n",A[0], A[1], A[2], A[3], A[4], A[5], A[6], A[7], A[8], A[9]);
    printf("%f, %f, %f, %f, %f, %f, %f, %f, %f, %f\n",(A[0]*C[0]), (A[1]*C[1]), (A[2]*C[2]), (A[3]*C[3]), (A[4]*C[4]), (A[5]*C[5]), (A[6]*C[6]), (A[7]*C[7]), (A[8]*C[8]
    printSum(A);

}
```

以矩陣作為參數並進行多種矩陣計算

(31).

```
fun main() {
  var a : int =3 ;
  {
      var a : int =3 ;
        {
          var a : int =3 ;
            {
              var a : int =3 ;
                {
                  var a : int =3 ;

                }
            }
        }
  }
  ret ;
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main (){
    int a = 3;
    {
        int a = 3;
        {
            int a = 3;
            {
                int a = 3;
                {
                    int a = 3;

                }
            }
        }
    }
return ;
}
```

巢狀 scoping 並接受回傳空值

(32). 簡單的矩陣轉置的測試

原始碼

```
fun transpose(A: int[3][3]){
    var S : int[3][3] = A ;

    println("matrix A is: ") ;
    println(A) ;


    for(var i : int=2; i+1; i=i-1){ //let i fom 2 to 0
        for(var j : int=2; j+1; j=j-1){ //let j fom 2 to 0
            A[i][j] = S[j][i] ;
        }
    }

    println("matrix of transpose A is: ") ;
    println(A) ;
    ret ;
}
fun main() : int {
    var A : int[3][3] = {1,2,3,4,5,6,7,8,9} ;
    transpose(A) ;

    println("transpose is a pointer operation, call function will pasaed it\'s pointer and change value like this:");
    println(A) ;

    ret 0 ;
    //1 2 3      1 4 7
    //4 5 6 -->  2 5 8
    //7 8 9      3 6 9
}
```

編譯出的 C

```
#include<stdio.h>
#include<stdlib.h>
void transpose (int A[3][3]){
    int S[3][3] = {A[0][0] , A[0][1] , A[0][2] , A[1][0] , A[1][1] , A[1][2] , A[2][0] , A[2][1] , A[2][2]};
    printf("%s\n","matrix A is: ");
    printf("%d, %d, %d, %d, %d, %d, %d, %d, %d\n",A[0][0], A[0][1], A[0][2], A[1][0], A[1][1], A[1][2], A[2][0], A[2][1], A[2][2]);
    for(int i = 2;(i+1);(i=(i-1))){
        for(int j = 2;(j+1);(j=(j-1))){
            (A[i][j]=S[j][i]);

        }

    }
printf("%s\n","matrix of transpose A is: ");
    printf("%d, %d, %d, %d, %d, %d, %d, %d, %d\n",A[0][0], A[0][1], A[0][2], A[1][0], A[1][1], A[1][2], A[2][0], A[2][1], A[2][2]);
    return ;

}
int main (){
    int A[3][3] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9};
    transpose(A);
    printf("%s\n","transpose is a pointer operation, call function will pasaed it\'s pointer and change value like this:");
    printf("%d, %d, %d, %d, %d, %d, %d, %d, %d\n",A[0][0], A[0][1], A[0][2], A[1][0], A[1][1], A[1][2], A[2][0], A[2][1], A[2][2]);
    return 0;

}
```

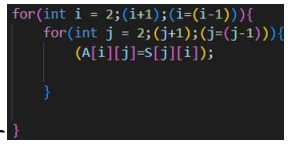
C 檔執行的結果

```
matrix A is:
1, 2, 3, 4, 5, 6, 7, 8, 9
matrix of transpose A is:
1, 4, 7, 2, 5, 8, 3, 6, 9
transpose is a pointer operation, call function will pasaed it's pointer and change value like this:
1, 4, 7, 2, 5, 8, 3, 6, 9
```

先宣告一個回傳為空值並以 3X3 int 矩陣為參數的函數 transpose，所以會生成對映的`void transpose (int A[3][3]){`的函數宣告，接著將 A 賦值給 S，對於 qv 的矩陣賦值轉成 C 後會一個個賦值`int S[3][3] = {A[0][0] , A[0][1] , A[0][2] , A[1][0] , A[1][1] , A[1][2] , A[2][0] , A[2][1] , A[2][2]};`而 println 也除了可以印自串也可以印矩陣，經轉換後也是一個個印出來

```
printf("%d, %d, %d, %d, %d, %d, %d, %d, %d\n",A[0][0], A[0][1], A[0][2], A[1][0], A[1][1], A[1][2], A[2][0], A[2][1], A[2][2]);
```

而 2 層的 for-loop 對矩陣元素進行交換，但因為沒有比較符號可用所以用 i+1 與 j+1 讓 loop 可以從 2→0 執行下去



，執行後因為回傳 void，所以轉成 `return ;` ，接著進入主函數，主函數要回傳 3int 值，而進入後會宣告 3X3 矩陣 A，此時會翻譯成 C 可接受的一維陣列初始化

```
int A[3][3] = {1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9};
```

之後也是可以接受矩陣 A 作為輸入 `transpose(A);` 並印出轉置前後的矩陣與位址操作對函數外的函數進行改變的結果，最後回傳 0 `return 0;` 代表主函數執行完成。以上就是簡單的範例，雖然編譯的程式碼不太美麗但可以保證結果是正確的，這也是本作業我覺得比較遺憾的地方了。