CS2433 Homework 1

CS18BTECH11001

February 23, 2020

This document is generated by LaTeX

Chapter 6 : Control Flow

Check your Understanding (CYU)

17. Why do most languages leave unspecified the order in which the arguments of an operator or function are evaluated?

A. *Most of the languages will let the compiler choose the order in the arguments of an operator or function to be evaluated, because the compiler will choose the order in which the results will be faster.*
Example :
y=(z+3);*
*x=y+*z;*
In this case the compiler will choose right to left order as the left to right order would have to load the value of y.
So the order will be unspecified in which the arguments evaluation been done.

18. What is short-circuit Boolean evaluation? Why is it useful?

A. *Circuit Boolean Evaluation : A compiler while doing boolean evaluation, will skips the second half of the computation if the overall value can be determined from the first half by generating a special code.This is called short-circuit Boolean evaluation.*

Uses :

i. It can be used to avoid out of bound subscripts
ii. It can reduce some amount of computation time

12. Given the ability to assign a value into a variable, why is it useful to be able to specify an initial value?

A. *For any variable that is declared with an initial value, it will be preallocated by the compiler in global memory, which will avoid the time of assigning during runtime.We can redue the programming errors that are caused due to the use of uninitialised variables if we specify an initial value.*

2. What distinguishes operators from other sorts of functions?

A. *A function consists of a name and a parenthesized list of arguments seperated by a comma where as an operator are in-built functions that use special type of syntax. Operators are simpler and they take 1 or 2 arguments which then dispenses them with parenthesis and commas.*

28. Why do most languages not allow the bounds or increment of an enumeration-controlled loop to be floating-point numbers?

A. *Comparision of floating point numbers do invlove round-off errors. As the no. of iterations depend on the comparision of flopating point numbers which leads to an unpredictable behaviour most of the languages donot allow enumeration-controlled loop to be floating-point numbers.*

16. Why is it impossible to catch all uses of uninitialized variables at compile time?

A. *As the variables that are allocated in stack/heap at runtime must be initialised at runtime, it is impossible to catch all the uses of these variables(uninitialised) at the compile time.*

Exercises

6.8 ***Languages that employ a reference model of variables also tend to employ automatic garbage collection. Is this more than a coincidence? Explain.***

A. *Reference count(RC) of an object is the number of references to that object. Initially, an object has 1 reference(ie., RC=1) and it will be incremented when a reference is created to it and decremented when a reference is destroyed. An object is said to be Garbage when there are no references to it(ie., RC=0). As the language that employ a reference model can identify the garbage using this reference count, most of them temd to employ automatic garbage collection.*

6.1 ***We noted in Section 6.1.1 that most binary arithmetic operators are left- associative in most programming languages. In Section 6.1.4, however, we also noted that most compilers are free to evaluate the operands of a binary operator in either order. Are these statements contradictory? Why or why not?***

A. *No, they are not contradictory. When we have identical operators Associativity just determines which subexpressions are arguments of which operator but it doesn't determine which subexpression would be evaluated first.*
Example : *For expression f(a)-g(a)-h(a) left associativity just groups as (f(a) - g(a)) - h(a) but it cannot decide whether f to be called first or g to be called first.*

6.24 ***Rubin [Rub87] used the following example (rewritten here in C) to argue in favor of a goto statement:***

```
int first_zero_row = -1;
/* none */
int i, j;
for (i = 0; i < n; i++) {
for (j = 0; j < n; j++) {
if (A[i][j]) goto next;
}
first_zero_row = i;
break;
next: ;
}
```

The intent of the code is to find the first all-zero row, if any, of an $n \times n$ matrix. Do you find the example convincing? Is there a good structured alternative in C? In any language?

A. *No, The above example is not convincing as the goto is used here which might lead to potential errors in other instances.*
The code can be re-written as :

```
int first _ zero _ row = -1;
/* none */
int i, j;
for (i = 0; i < n; i++) {
  flag = 0;
  for (j = 0; j < n; j++) {
    if (A[i][j]) {
      flag = 1;
      break;
    }
  }
  if(flag==0) {
  first _ zero _ row = i;
  break;
  }
}
```

6.7 Is & (&i) ever valid in C? Explain.

A. *The Unary operator & needs an lvalue(expression which can be placed on the left-hand side of the assignment operator) as it's operand and return a rvalue(expression which can be placed on the right-hand side of the assignment operator). As & i is an rvalue it can't be an operand for &. So &(&i) is not valid in C.*

6.4 Translate the following expression into postfix and prefix notation:

$$[-b + sqrt(b \times b - 4 \times a \times c)]/(2 \times a)$$

Do you need a special symbol for unary negation?

A. *Prefix notation* : / + ∼ b sqrt - × b b × 4 × a c × 2 a
Postfix notation : b ∼ b b × 4 a × c × - sqrt + 2 a × /
Yes, We need a special symbol to represent unary negation.

Chapter 7 : Types

Check your Understanding (CYU)

2. What does it mean for a language to be strongly typed? Statically typed? What prevents, say, C from being strongly typed?

A. *For a language to be strongly typed, if for a given program, the data types of all constants and variables must be predefined as part of the programming language.*
Statically typed is a programming language characteristic in which variable types are explicitly declared and thus are determined at compile time.
Strongly typed languages have restrictions on conversions between types. Variant records or Union records prevent C from being strongly typed as there we can't check which varint is present in the variant record. And also you can convert any type to any other type through a cast, without a compiler error.So, it is not considered to be strongly typed.

7. Give two examples of languages that lack a Boolean type. What do they use instead?

A. *Lisp, C are examples of languages that lack a Boolean type.*
Lisp uses an empty list for false and any other values for true.
C uses 0 for false and nonzero integer for true.

10. What are aggregates?

A. *Aggregates is a type of data which is considered as a single entity but has more than one piece of data. These are used to group all the related data together. Array, Classes, Structures are some of the examples.*

11. What is the difference between type equivalence and type compatibility?

A.

Type equivalence	***Type compatibilty***
The object A is said to be type equivalent to object B only if data type of object A is same as data type of object B	*The object A is said to be type compatible to object B if they are equivalent or if type of A can be coerced to type of B*
Equivalence is a tighter relationship	*Compatibilty is a looser relationship*
Types must be equal	*Types may be different by one has to be convertable to other*

15. Explain the differences among type conversion, type coercion, and nonconverting type casts.

A. *Type conversion refers to changing an entity from one datatype into other.*
Implicit type conversion is called type coercion.
The Changes of type where the underlying bits do not alter are called nonconverting type casts.

19. What is type inference? Describe three contexts in which it occurs.

A. *The detection of the data type of an expression automatically in a programming language is reffered as type inference.*
Context of type inference :
C++ : auto keyword triggers type inference
Python : variables declared with var triggers a limited form of type inference
Haskell : annotations for functions are also used to check that type inference actually found the types you expected.

23. Why is it easier to implement assignment than comparison for records?

A. *Assignments can occur in a single operation. Comparisons can be performed line by line, but comparing whole blocks can fail on records with different garbage data in the holes. indicates the compiler should optimize for space instead of performance.*

26. Briefly describe two purposes for unions/variant records.

A. *Purposes for Union :*

- *i. It provides an efficient way of reusing the memory location. This is due to the property that only one of its members can be accessed at a time.*
- *ii. Safer and more portable nonconverting casts can be achieved*
- *iii. unions are used to represent alternative sets of fields within a record.*

33. Discuss the comparative advantages of contiguous and row-pointer layout for arrays.

A. *Contiguous layout is typically more space efficient, but may be slower on machines with slow multiplies,and may be less space efficient when rows have different lengths.*
A row-pointer layout allows easy construction of new arrays from pre-existing rows from possibly multiple arrays without copying data.

34. Explain the difference between row-major and column-major layout for contiguously allocated arrays. Why does a programmer need to know which layout the compiler uses? Why do most language designers consider row-major layout to be better?

A. *In a row-major order, the consecutive elements of a row reside next to each other, whereas In a column-major order, the consecutive elements of a column reside next to each other.*
If the compiler uses row-major layout, then iterating in row-major order may be faster than iterating through its elements in column-major order. So the programmer need to know which layout the compiler uses for the faster implementation.
As reading memory in contiguous locations is faster than jumping around among locations due to caching, most language designers consider row-major layout to be better.

42. Discuss the advantages and disadvantages of the interoperability of pointers and arrays in C.

A. *Advantages :*

- *i. Due to this property, if we know know a pointer to an element in array we can also know the element that is at a k positions from the given element by a simple addition.*
- *ii. Pointers can be compared for ordering or can be subtracted from one another, provided that they refer to elements of the same array.*

Disadvantages :

- *i. For a two-dimensional array with contiguous layout, the formal parameter may be declared as int a[ ][m] or int (*a)[m]. The size of the first dimension is irrelevant; all that is passed is a pointer, and C performs no dynamic checks to ensure that references are within the bounds of the array.*
- *ii. In most cases, sizeof can be evaluated at compile time; the principal exception occurs for variable-length arrays, whose shape is not known until elaboration time.*

45. What is garbage? How is it created, and why is it a problem? Discuss the comparative advantages of reference counts and tracing collection as a means of solving the problem.

A. *The objects, data, or other regions of the memory of a computer system (or other system resources), which will not be used in any future computation by the system, or by a program running on it is said to be called as Garbage.*
When we write programs, we often use variables that are not referenced anymore. These use memory space but are not used in the program. This turns out to be garbage. As the time goes, the no. of garbage values grow bigger and there may be a chance of memory leak if they aren't cleared.
Reference count are able to find the garbage(RC=0) and then they can be easily cleared.When reference counts are non zero, tracing collectors work by recursively exploring the heap, starting from external pointers, to determine what is useful.
Inability of the RC to handle cycles and the tendency of the tracing collection to stop the world periodically in order to reclaim space. Tracing generally requires a reversed pointer indicator in every heap block, which reference counting does not.

46. Summarize the differences among mark-and-sweep, stop-and-copy, and generational garbage collection.

A. *mark-and-swap :*
Collect garbage in two phases:
The Mark Phase : it visits all reachable objects and marks them as visited.
The sweep phase : it sweeps through all objects in memory, adding those not marked to the free list.
Mark-and-sweep collection has the disadvantage that collection overhead is proportional to the size of memory, which can be large in modern Lisp systems.
stop-and-copy :
Copying collection divides the heap into semispaces, and copies reachable objects between semispaces during collection. Because only reachable objects are visited, the overhead of copying collection is no longer proportional to the size of memory. Copying collection has the further advantage that reachable objects are placed contiguously when copied and thus are compacted.
Generational garbage collection :
Generation garbage collection is a technique that focuses on the garbage collection on the youngest objects because they are most likely to become garbage. The collection references are localized and garbage collection does not disrupt the reference locality of the program as much. Collection is less likely to disrupt interactive users.

Exercises

7.6 Ada provides two remainder operators, rem and mod for integer types, defined as follows [Ame83, Sec. 4.5.5]: Integer division and remainder are defined by the relation A = (A/B)*B + (A rem B), where (A rem B) has the sign of A and an absolute value less than the absolute value of B. Integer division satisfies the identity (-A)/B = -(A/B) = A/(-B). The result of the modulus operation is such that (A mod B) has the sign of B and an absolute value less than the absolute value of B; in addition, for some integer value N, this result must satisfy the relation A = B*N + (A mod B). Give values of A and B for which A rem B and A mod B differ. For what purposes would one operation be more useful than the other? Does it make sense to provide both, or is it overkill? Consider also the designers of these languages could have picked semantics resembling those of either Adas rem or its mod.Which did they pick? Do you think they made the right choice?

A. *As the sign of rem is dependent on the sign of dividend and sign of mod is dependent on the sign of divisor, A rem B and A mod B differ when A(dividend) and B(divisor) have different signs. The operator rem is used to find out the remainder when A is divided by B where as the operator mod is used to find out the value from which A must be substracted such that the resultant value can be divisible by B(congruency). It is not compulsory to provide both of them but when we look into correctness we need both of them. So, it is not a overkill. When considered in the designers point of view they have picked 'rem' operator as it is most commonly used. Yes, I think they made a right choice.*

7.15 Consider the following C declaration, compiled on a 32-bit Pentium machine:
struct {
int n ;
char c ;
} A[10][10] ;
If the address of A[0][0] is 1000 (decimal), what is the address of A[3][7]?

A. *Size of int in 32-bit Pentium machine = 4B*
Size of char in 32-bit Pentium machine = 1B
But, due to padding the size of struct becomes = 4+4 = 8B
Given, the address of A[0][0] = 1000
Then the address of A[3][7] will be $= 1000 + 8 \times (3 \times 10 + 7)$
$= 1296$

7.17 Suppose A is a 10×10 array of (4-byte) integers, indexed from[0][0] through [9][9]. Suppose further that the address of A is currently in register r1, the value of integer i is currently in register r2, and the value of integer j is currently in register r3. Give pseudo-assembly language for a code sequence that will load the value of A[i][j] into register r1 (a) assuming that A is implemented using (row-major) contiguous allocation; (b) assuming that A is implemented using row pointers. Each line of your pseudocode should correspond to a single instruction on a typical modern machine. You may use as many registers as you need. You need not preserve the values in r1, r2, and r3. You may assume that i and j are in bounds, and that addresses are 4 bytes long. Which code sequence is likely to be faster? Why?

A. a) *Using row-major contiguous allocation* :

```
r2 ×:= 40
r3 << := 2
r1 +:= r3
r1 +:= r2
r1 := *r1
```

b) *Using row pointers* :

```
r2 << := 2
r1 +:= r2
r1 := *r1
r3 < < := 2
r1 +:= r3
r1 := *r1
```

We can observe that when we use row-mojor allocation it has completed in 5 steps where as for the row pointers it has to perform one more load operation.

7.20 Explain the meaning of the following C declarations:
double a[n];
double (b)[n];
double (c[n])();
double (d())[n];

A. i. *Array of n pointers to doubles*
ii. *Pointer to array of n doubles*
iii. *Array of n pointers to functions returning doubles*
iv. *Function returning pointer to array of n doubles*

7.49 Learn about weak references in Java. How do they interact with garbage collection? Describe several scenarios in which they may be useful.

A. *Java weak reference is used to release unused objects for Garbage collector(GC) very efficiently. When there are one or more reference to an object it will not be garbage collected in Java. But this rule depends on what type of reference it is. If an object has only weak reference associated with other objects, then it is a valid candidate for garbage collection.*

Uses :

i. Weak references are used to implement canonicalizing mappings(mappings where only one instance is hold for a particular value). Instead of creating a new object, it looks for the existing one in the mapping and uses it.

ii. Weak references are mostly used in WeakHashMap class. It is a map implementation where every key is stored as a weak reference to a given key. When the GC removes a key, the entity associated with this key is deleted as well.