Examples for Programming Language Features

Ch 1 Preliminaries

1.3 Language Evaluation Criteria

1.3.3 Reliability

1.3.3.1 Type Checking

The following C program compiles and runs!

```c
foo (float a) {
    printf ("a: %g and square(a): %g\n", a, a*a);
}
main () {
    char z = 'b';
    foo(z);
}
```

Output is: **a: 98 and square(a): 9604**

*Subscript range* checking for arrays is a part of type checking, but it must be done in the **run-time**. Out-of-range subscript often cause errors that do not appear until long after actual violation.

```c
#include <stdio.h>
foo (char s[])  {
    printf("5th char in %s is %c\n", s, s[4]);
}
main () {
    char str[] = "abc";
    foo(str);
}
```

produces:

*5th char in abc is*}

Ch 3. Describing Syntax and Semantics

3.1. Formal Methods of Describing Syntax

Example: Python formal definition

(https://docs.python.org/3/reference/simple_stmts.html)
Ch 5. Names, Bindings, and Scopes

5.4.2.1 Static Type Binding

In Perl, $x is a scalar, @x is an array, %x is a hash.

```perl
$x = 5;
print $x . "\n";
$x = "Hello";
print $x . "\n";
@x = ("Ali", "Ayse", "Can", "Canan");
print "$x \n";
print "@x \n";
%
data = ('A', 4.0, 'A-', 3.7, 'B+', 3.3, 'B', 3);
print "$data{A-}\n";
```

5.4.2.2. Dynamic Type Binding

Javascript uses dynamic type binding.

```javascript
list = [2, 4.33, 6, 8]; list is a single dimensioned array.
list = 73 list is a simple integer.
```

5.4.3.1. Static Variables

Ex in C:

```c
int a = 5; // global static variable
int foo()
{
    static int b = 10; // static variable, scope is only the function.
}
```

5.4.3.3. Explicit Heap-Dynamic Variables

Ex in C++

```c++
int *intnode; // create a pointer
intnode = new int; // create the heap-dynamic variable
delete intnode; // deallocate the heap-dynamic variable
```
Ex in Java:
   Integer i = new Integer(5);

5.4.3.4 Implicit Heap-Dynamic Variables

Example in JavaScript:

numbers = [3, 5, 7, 15];

5.4-Summary

C Example:

int a1[10];                // static global variable.
static int a2 = 5;          // static global variable, initialized
void foo() 
{
    static int a3[10];       // static local array variable
    int a4[10];               // stack-dynamic variable
    int *a5 = (int *)malloc(10*sizeof(int));
    // a5 (pointer) is stack-dynamic
    // *a5 is explicit heap-dynamic

    ...  
    free(a5);  
}

5.5.6. Dynamic Scope

function big() {
   var x: integer;
   function sub1() {
      ... ?
      ...
   }
   function sub2 () {
      var x: integer;
      ...
      sub1 ;
   }

   Sub2();
   sub1();
}
5.8 Named Constants
Java and C++ allow dynamic binding of values to named constants.

Example in Java

```java
import java.util.Scanner;
public class A {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        int a = kb.nextInt();
        final int b = a * 2;
        System.out.println(a+ " "+b);
    }
}
```

6. Data Types

6.3.3 String Length Options

- **Limited dynamic length strings**: (C and C++)

Following are equivalent in C

```c
char c[] = "abcd";
char c[50] = "abcd";
char c[] = {'a', 'b', 'c', 'd', '\0'};
char c[5] = {'a', 'b', 'c', 'd', '\0'};
```

6.4 Enumeration Types

- **Example in C**:

  ```c
  /* enumeration type in C */
  main () {
    enum day {mo, tu, we, th, fr, sa, su};
    enum day today;
    today = th;
    if ((today == sa) || (today == su))
      printf ("weekend\n");
    else printf ("weekday\n");
  }
  ```

6.5.3 Subscript Bindings and Array Categories

- **Heap-dynamic array**: Similar to stack-dynamic array, but the size can change during the lifetime.
  
  Advantage: the array can grow and shrink.
Example:

ArrayList in Java, List in python, Arrays in perl (dynamic-array.pl)

Python

```python
>>> mylist = [2, 1.6, (3+5j), 'c', 'a word']
>>> print mylist
[2, 1.6, (3+5j), 'c', 'a word']
>>> x = (1+2j) * mylist[2]
>>> x
(-7+11j)
```

6.6 Associative Arrays

Provided in Perl (called hash) and Java.

* Indexed by keys
* The size can grow and shrink dynamically.

```perl
#!/usr/local/bin/perl
%notlar = ("Ali" => 95, "Veli" => 80, "Selami" => 95);
To print an entry:

    print "Veli'nin notu $notlar{Veli}\n";
To add a new entry:

    $notlar{"Hulki"} = 75;
To delete an entry:

    delete $notlar{"Veli"};
To change an entry:

    $notlar{"Veli"} = 82;
To go through the array:

    foreach $anahtar (keys %notlar) {
        print "$anahtar $notlar{$anahtar}\n";
    }
To empty the array:

    %notlar = ();
```

6.8. Tuple Types

A tuple is a data type that is similar to a record, except that the elements are not named.

Ex: Python

```python
>>> myTuple = (3, 5.8, 'apple')
>>> myTuple[1] 5.8
```
>>> myTuple[1] = 6.7
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> myTuple = [3, 5.8, 'apple']    # Now, it is a list
>>> myTuple[1] = 6.7
>>> myTuple = (3, 5.8, 'apple')  # Back to tuple
>>> yourTuple = (1, 'pear')
>>> print myTuple + yourTuple
(3, 5.8, 'apple', 1, 'pear')
>>> print yourTuple
(1, 'pear')
>>> del yourTuple
>>> print yourTuple
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'yourTuple' is not defined

List Types
Lists were first supported in Lisp and Scheme.

Ex: Python:

>>> myList = [1, 2.3, "three"]
>>> del myList[1]
>>> myList
[1, 'three']
>>> myList.insert(1,"orange")
>>> myList
[1, 'orange', 'three']
>>> myList.append("plum")
>>> myList
[1, 'orange', 'three', 'plum']

6.10. Union Types

A union type may store different type values at different times in the same location.

Type checking must be dynamic.

Example in C:
#include <stdio.h>
#include <string.h>
union Data {
    int i;
    float f;
    char str[20];
};

int main() {
    union Data data;
    data.i = 10;
    data.f = 220.5;
    strcpy( data.str, "C Programming" );
    printf( "data.i : %d\n", data.i );
    printf( "data.f : %f\n", data.f );
    printf( "data.str : %s\n", data.str );
    return 0;
}

Generates the following output:
data.i : 1917853763
data.f : 4122360580327794860452759994368.000000
data.str : C Programming

Size of a variable of union type:
#include <stdio.h>
#include <string.h>

union Data {
    int i;
    float f;
    char str[20];
};

int main() {
    union Data data;
    printf("Memory size occupied by data: %d\n", sizeof(data));
    return 0;
}

Prints 20.

6.11.3.2. Lost heap-dynamic variables

C++ Example

void foo() {
    int * x = new int;
}

foo(); // no way to reach the allocated integer: memory leakage
C++ Example

```cpp
int * x = new int;
x = new int; // no way to reach the first allocated integer: memory leakage
```

9. Subprograms

9.2.3 Parameters

Python Example / positional parameter passing

```python
def sum(x, y, z):
    return x + y + z
u = sum(a, b, c)
```

Python Example / positional with defaults

```python
def sum(x, y, z=3):
    return x + y + z
u = sum(a, b)
u = sum(a, b, c)
```

Python Example / keyword based:

```python
def sum(x, y, z=3):
    return x + y + z
u = sum(y=a, x=b)
u = sum(y=a, z=c, x=b)
```

Python Example / variable number of arguments:

```python
def sum(x, y, *z):
    s = x + y
    for e in z:
        s = s + e
    return s
u = sum(a, b)
u = sum(a, b, c, 3, 5)
```

Python Example / variable number of arguments:

```python
def sum(x, y, *z, **w):
    s = x + y
    for e in z:
        s = s + e
    for k,e in w.iteritems():
        s = s + e
    return s
u = sum(a, b)
u = sum(a, b, c, d, i=3, j=5)
```
Python Example / unpacking:

```python
def sum(x, y, z, t):
    return x + y + z + t

v = [3, 5, 6, 8]
u = sum(*v)  # same as sum(3, 5, 6, 8)
```

9.5.2 Parameter passing methods

The output of the program considering static scoping, sub-expression evaluation order of left-to-right, and:

- parameter passing with pass-by-value-result
- parameter passing with pass-by-reference
- parameter passing with pass-by-value

```c
int y;
int z;
int foo(int x) {
    x = x + 5;
    y = x;
    x = x + 6;
    return y;
}

int bar(int x) {
    y = foo(x) * x;
    return x;
}

void main() {
    y = 4;
    z = bar(y);
    print(y, z);
}
```

// Pass-by-value-result: 15 15
// Pass-by-reference: 225 225
// Pass-by-value: 36 4

9.5.2.5. Pass-by-name

**Example in Scala**

[https://alvinalexander.com/source-code/scala/simple-scala-call-name-example](https://alvinalexander.com/source-code/scala/simple-scala-call-name-example)

(Scala is a type-safe JVM language that incorporates both object oriented and functional programming)
9.6 Parameters That Are Subprograms

Python Example

```python
def map(op, items):
    res = []
    for item in items:
        res.append(op(item))
    return res

def square(x):
    return x*x

def triple(x):
    return 3*x

print(map(square, [1, 2, 3]))  # prints [1, 4, 9]
print(map(triple, [1, 2, 3]))  # prints [3, 6, 9]
```

C++ Example

```cpp
vector<int> map(int (*op)(int), vector<int> const & list){
    vector<int> res;
    for (int v: list){
        int mappedValue = (*op)(v);
        res.push_back(mappedValue);
    }
    return res;
}

int square(int x){
    return x*x;
}

int triple(int x){
    return 3*x;
}

vector<int> list = {1, 2, 3, 4};
vector<int> res = map(&square, list);
vector<int> res2 = map(&triple, list);
```

9.12. Closures

A JavaScript closure example:

http://dijkstra.cs.bilkent.edu.tr/~guvenir/closure.htm
Python Example:

def make_adder(x):
    def adder(y):
        return x+y # x is captured from y.
    return adder # make_adder returns inner function.

f = make_adder(3) # f is a function
print(f(5)) #prints 8
# 3 of the previous line is captured as x

C++ Example:

function<int(int)> make_adder(int x){
    auto adder = [=] (int y) {return x+y; };
    return adder;
}
auto adder = make_adder(3);
cout << adder(5);

9.13. Coroutines

Not exactly a full-coroutine implementation, but a similar feature in Python, “generators” example:

def genPairs(n):
    for i in range(0, n):
        for j in range(i+1, n):
            yield (i, j)

for p in genPairs(5):
    print(p)
10. Implementing Subprograms

10.2 Implementing “Simple” Subprograms

Example in Fortran

```fortran
INTEGER I, J
COMMON I
CALL X
GOTO 10
10 CONTINUE
END

SUBROUTINE X
INTEGER K, J
COMMON I
K=5
I=6
J=I+K
RETURN
END
```

The translator performs the following bindings for the main program:

- \( I \leftrightarrow d[\text{COMMON, 0}] \)
- \( J \leftrightarrow d[\text{MAIN, 1}] \)
- \( 10 \leftrightarrow c[\text{MAIN, 3}] \) \hspace{1cm} \text{CONTINUE statement}
- \( X \leftrightarrow c[X, 0] \) \hspace{1cm} \text{first statement of X}

And the following bindings for the subroutine X:

- \( I \leftrightarrow d[\text{COMMON, 0}] \)
- \( K \leftrightarrow d[X, 1] \)
- \( J \leftrightarrow d[X, 2] \)

The linker then performs a further binding of addresses by combining the main program and the subroutine X.

It produces the following bindings in the main program:

- \( I \leftrightarrow D[0] \)
- \( J \leftrightarrow D[2] \)
- \( 10 \leftrightarrow C[3] \)
- \( X \leftrightarrow C[5] \)

And the following bindings for X:

- \( I \leftrightarrow D[0] \)
- \( K \leftrightarrow D[4] \)
- \( J \leftrightarrow D[5] \)
The contents of the memory will be as follows:

**Code memory**

- \( D[3] := ip + 2 \)
- \( ip := 5 \)
- \( ip := 3 \)
- noop
- halt

**Data memory**

- \( D[4] := 5 \)
- \( D[0] := 6 \)
- \( ip := D[3] \)
- \( ip := D[3] \)

The contents of the Data memory during the execution:
Static Chains

The most common way to implement static scoping in languages that support nested subprograms is static chaining.

A new pointer, called Static Link, is added to the Activation Record. **Static Link** is sometimes called **static scope pointer**.

Static Link is used to access to nonlocal variables. **Static Chain** is the chain of static Links.

When a reference is made to a nonlocal variable, the Activation Record instance is in the static chain.

How to find the ARI that contains the nonlocal variable?

Compiler can determine the distance of the ARI on the static chain.

**Static_depth** of a subprogram is the depth of nesting.

\[
\text{Static}\_\text{depth}(\text{main}) = 0.
\]

\[
\text{Static depth}(A) = 1 + \text{static}\_\text{depth}(A\text{'s static parent}).
\]

**Nesting_depth (chain_offset) of a variable:**

\[
\text{static}\_\text{depth}(\text{ARI of reference}) - \text{static}\_\text{depth}(\text{ARI of declaration})
\]

Then, address of a variable is represented by a pair

\[
(\text{chain}\_\text{offset}, \text{local}\_\text{offset}),
\]
function main()
  var x = 5;

function bigsub() {
  var a, b, c;
  function sub1 {
    var a, d;
    a = b + c;
    ...
  } // end of sub1
  function sub2(x) {
    var b, e;
    function sub3() {
      var c, e;
      ...
      sub1();
      ...
      e = b + a;
    } // end of sub3
    ...
    sub3();
    ...
    a = d + e;
  } // end of sub2
  ...
  sub2(7);
  ...
} // end of bigsub

... bigsub();
...
} // end of main
## Activation Records:

**main:**

- Local variable \((x)\)

**bigsu**

- Local var \((c)\)
- Local var \((b)\)
- Local var \((a)\)
- Dynamic Link
- Static Link
- Return Address

**local_offset(a) = 3**

**sub1:**

- Local var \((d)\)
- Local var \((a)\)
- Dynamic Link
- Static Link
- Return Address

**sub2:**

- Local var \((e)\)
- Local var \((b)\)
- Parameter \((x)\)
- Dynamic Link
- Static Link
- Return Address

**sub3:**

- Local var \((e)\)
- Local var \((c)\)
- Dynamic Link
- Static Link
- Return Address

```c
function main() {
```
var x = 5;
function bigsub() {
    var a, b, c;
    function sub1() {
        var a, d;
        a = b + c; ←--------- (0,3) = (1,4) + (1,5)
        ...
    } // end of sub1
    function sub2(x) {
        var b, e;
        function sub3() {
            var c, e;
            ...
            sub1();
            ...
            e = b + a; ←--------- (0,4) = (1,4) + (2,3)
        } // end of sub3 ...
        sub3();
        ...
        a = d + e; ←--------- (1,3) = d: ??? Error
    } // end of sub2
    ...
    sub2(7);
    ...
} // end of bigsub
...
bigsub();
...
} // end of main

Contents of the Runtime Stack:

Top
ARI of main
EP

l.v. x=5
in main()
**How to maintain the static links?**

Where should a Static Link point to when a new ARI is created?

It should point to the most recent ARI of the static parent.

**How to find the most recent ARI of the static parent?**

For example, when `sub1()` is called from `sub3()`, the ARI of `sub1()` should point to the ARI of `bigsub()`, which is the static parent of `sub1()`.

**Solution:** Compiler computes: \( \text{nesting\_depth(caller, called)} \): as

\[ \text{Static\_depth(caller)} - \text{static\_depth (unit that declared called)}. \]

It the example above:

- Caller: `sub3`
- Callee: `sub1`
- Static parent of Callee: `bigsub`

\[
\text{nesting\_depth(sub3, sub1)}:
= \text{Static\_depth(sub3)} - \text{static\_depth(unit that declared sub1)}
= \text{Static\_depth(sub3)} - \text{static\_depth(bigsub)}
= 3 - 1
= 2.
\]

So, SL of the ARI of `sub1` should point to the 2\(^{nd}\) ARI in the static chain.