

CSCI 3155 – Worksheet #1

September 18, 2003

Three things to keep in mind:

- The questions on this worksheet will NOT be graded. You are, however, more than welcome to discuss your individual results during office hours!
- Your results on this worksheet do NOT affect your grade in any way.
- Please do not consider this worksheet to be sufficient as a preparation for the first exam; it is merely a device intended to point you towards some of the things you might have missed.

1 Types

1. Consider the following recursive structural type definitions:

```
struct A {  
    struct B *x;  
    struct C *y;  
};
```

```
struct B {  
    int x;  
    struct B *y;  
};
```

```
struct C {  
    int x;  
    struct A *y;  
};
```

Which, if any, of them are structurally equivalent?

2. Consider the following recursive structural type definitions:

```
TYPE T = RECORD  
    a : INTEGER;
```

```
    b : S
END;

TYPE S = RECORD
    b : T;
    a : INTEGER
END;
```

(in this MODULA-3 definition, “RECORD . . . END” take the place of “struct { . . . }” in C, and all uses of record types turn the respective variable into a pointer/reference implicitly.) Are they equivalent according to MODULA-3’s notion of structural type equivalence?

3. Type constructors can be thought of as “types with parameters”. How many parameters does MYSTERY’s ARRAY type constructor have?
4. In programming languages with support for associative arrays, it is often possible to construct associative arrays which take multiple indices (of varying types) and return arbitrary return types.

From the point of view of a programmer, what are the differences between associative arrays (with an arbitrary number of indices) and functions?

For this question, assume that the functions we are considering do not have any externally visible side effects. Explain.

2 Scopes

For this part, consider the following program:

```
PROCEDURE P(x : INTEGER) : INTEGER
  PROCEDURE Q(y : INTEGER) : INTEGER
  BEGIN
    RETURN x + y;
  END
  PROCEDURE R() : INTEGER
  VAR x : INTEGER;
  BEGIN
    x := 7;
    RETURN Q(2);
  END
BEGIN
  IF x < 0 THEN
    RETURN Q(0);
  ELSE
    RETURN R();
  END
END
```

1. What does P(-1) return if we are using static scoping?
2. What does P(1) return if we are using static scoping?
3. What does P(-1) return if we are using dynamic scoping?
4. What does P(1) return if we are using dynamic scoping?

3 Storage and Binding

1. Which attributes constitute a variable?
2. Describe, in your own words, what heap-dynamic allocation is.
3. Some programming languages implement *Garbage collection*, a mechanism which automatically frees memory that is no longer in use. Garbage collection does not, however, consider any memory for which, at compile time, a deallocation site is already known.

Which of the four types of storage binding can make use of Garbage Collection?

4. In C, the keyword 'static' allows a local variable (a variable declared within a block) to have static binding. Consider the following program:

```
int
foo(int x)
{
    static int k = x;

    if (x == 0)
        return 0;

    if (x < 0)
        return k + foo(-x);
    else {
        int y = foo(x-1);
        return k + y + 1;
    }
}
```

What will this program compute?

4 Parsing

1. Describe, in your own words what a *Lexeme* is.
2. Consider the following grammar:

$$\begin{aligned}\langle S \rangle &\longrightarrow \langle A \rangle \\ &\quad | (\langle B \rangle) \\ \langle A \rangle &\longrightarrow \langle B \rangle . \langle B \rangle \\ \langle B \rangle &\longrightarrow \text{id } (\langle A \rangle) \\ &\quad | \langle A \rangle [\langle C \rangle] \\ \langle C \rangle &\longrightarrow \langle B \rangle + \langle B \rangle \\ &\quad | \langle B \rangle - \langle B \rangle \\ &\quad | \langle B \rangle\end{aligned}$$

What are its productions, i.e., what is the language it generates?

3. Consider the following grammar:

$$\begin{aligned}\langle S \rangle &\longrightarrow \text{id } \langle A \rangle \\ \langle A \rangle &\longrightarrow \text{number } \langle A \rangle | \epsilon\end{aligned}$$

While trying to put this into a parser generator, you realise that the generator does not support ϵ rules. Try to rewrite it so that you get rid of the ϵ while still recognising the same grammar.

5 Miscellaneous

1. Implement a PROCEDURE `fac(x : INTEGER) : INTEGER` in MYSTERY which computes the factorial of the integer passed to it (for this task, assume that `x` is nonnegative). For this question, assume that MYSTERY's settings do NOT allow you to make use of the assignment operator (`:=`).
2. Assume that you are using a programming language like C++, but *without* the following features:
 - Arrays (including `new[]`)
 - C memory management features (like `malloc()`, `calloc()`, `realloc()` and friends)
 - library classes and library functions

Functions and classes as concepts are still permitted, as are pointers and the use of `new` for object allocation.

- (a) Implement a class `QuasiArray` with the following member functions:
 - `long get(int index);`
 - `void put(int index, long value);`

such that, for an object `qa` of type `QuasiArray`, `qa.get(i)` behaves like a read operation on an array at index `i`, and `qa.put(i, v)` behaves like a write operation for value `v` at index `i`.

- (b) Explain the advantages and disadvantages of your implementation over fixed and non-fixed heap-dynamic arrays.