

CS2104 Tutorial 5: Data Representation II

1. In C, the declaration `int x` would mean that “the type of `x` is an integer”; the declaration `int *x` would mean that “the type of `x` is a pointer to an integer”; the declaration `int x() {...}` would mean that “`x` is a function return an integer type”. What is the type of `x` if `x` is declared as:

```
int *(*x)[6]();
```

2. Kernighan and Ritchie (the designers of C) wrote “*The C Programming Language*” (published by Prentice Hall). The first edition in 1978 was *before* the standardization of C¹. When describing conversion rules for arithmetic expressions, such as $x + y$, they wrote:

First any operands of type `char` or `short` are converted to `int`, and any of type `float` are converted to `double`. Then, if either operand is `double`, the other is converted to `double` and that is the type of the result.

Otherwise, if either operand is `long`, the other is converted to `long` and that is the type of the result.

Otherwise, if either operand is `unsigned`, the other is converted to `unsigned` and that is the type of the result.

Otherwise both operands must be `int`, and that is the type of the result.

- (a) Based on the above information, list down all the type rules needed for a type system on arithmetic expressions.
- (b) Use the type system that you defined to type check the following expression `'0' + 1.0 * (-1 + x)` (assuming `x` is an `unsigned` of value 1). Hence state (i) the type of the result; and (ii) the value of the result.

3. Here is a code segment consisting of type and variable declarations:

```
type
  range = -5..5;
  table1 = array [range] of char;
  table2 = table1;
var
  x,y : array [-5..5] of char;
  z   : table1;
  w   : table2;
  i   : range;
  j   : -5..5;
```

State which variables are type equivalent under (i) Structural Equivalence; (ii) Name Equivalence; (iii) Declaration Equivalence, using the appropriate equivalence rules.

4. Write a procedure in Pascal

```
procedure GIGO(I:integer; var R:real)
```

that uses a record with two variants, whose only purpose is to compromise the type-checking system. `GIGO` takes an argument `I` of type *integer* and returns the same bit pattern as a result `R` of type *real* without actually converting the value of `I` to a real number

¹The second edition in 1988 was for ANSI Standard C.