David W. Morris
Professor Eileen Head
Programming Languages – Homework 9
Due: April 1, 2015

1)

Macros are potentially Faster, take up less space, and do not require type-checking.
Functions force evaluation of variables, so they are slower, take up more space than macros, and require type-checking.
Inline Functions are between Functions and Macros in terms of speed, take up the most amount of space, and require type-checking.

Macros are faster whenever there is a situation where an expression might not need to be evaluated.
Functions will be faster when there is a situation where all expressions of the function must be evaluated for the correct result to be reached every time it is run.

2)

Assuming that each of these assignments is happening in a vacuum and are unaffected by each other, any assignments where either vp or ip are on the right side of the assignment are invalid, because these two have not been instantiated yet. *ip will have an output of 3 after the final assignment, ip = reinterpret_cast<int*>(dp); , because it is the only assignment that correctly translates the double* type to an int*.

3)

double *a[n]; → A is array of n pointer to double

Double (*b)[n]; → B is pointer to array of n double

double (*c[n])(); → C is array of n pointer to function returning double

double (*d())[n]; → D is function returning pointer to array of 7 double

4)

i+1 = 4 → 0 + sizeof(int*)*1
i+k = 20 → 0 + sizeof(int*)*k
k = 5 → 20 / sizeof(int*)
a+k = 100 → 0 + sizeof(A*)*5
&(a[9]) = 180 → 0 + sizeof(A)*9
&(a[9]) – (a+1) = 8 → I know it has something to do with sizeof(int*) = 8, but I have no idea why it does this.

5)

Value: (5 + 7) * 2 == 24
Macro: 5 + 7 * 2 == 19

6)

Value:
{3,2,1,0}

Reference:

f(A[1],A[1])

{ 3, 5, 1, 0 }

Value-Result:

{ 3, 6, 1, 0 }

Macro Expansion:

A[i++](I=1) = A[i++] (I = 2)- 1

A[i](I=2) = A[i](I=2) + A[i++](I=3) + A[0]

{ 3, 0, 4, 0 }

Name:

{ 3, 5, 1, 0 }

7)

The representation of Bar holds two fields for the b field, however only the one defined in class bar is accessible.

8)

This is acceptable in C++ because pointers are a variable type that can be assigned to anything, while functions are pre-defined and bound at compile time. This is not an issue in Java because the programmer has no control nor access to pointers, and java processes things like this as references.