Assignment 4

CS314

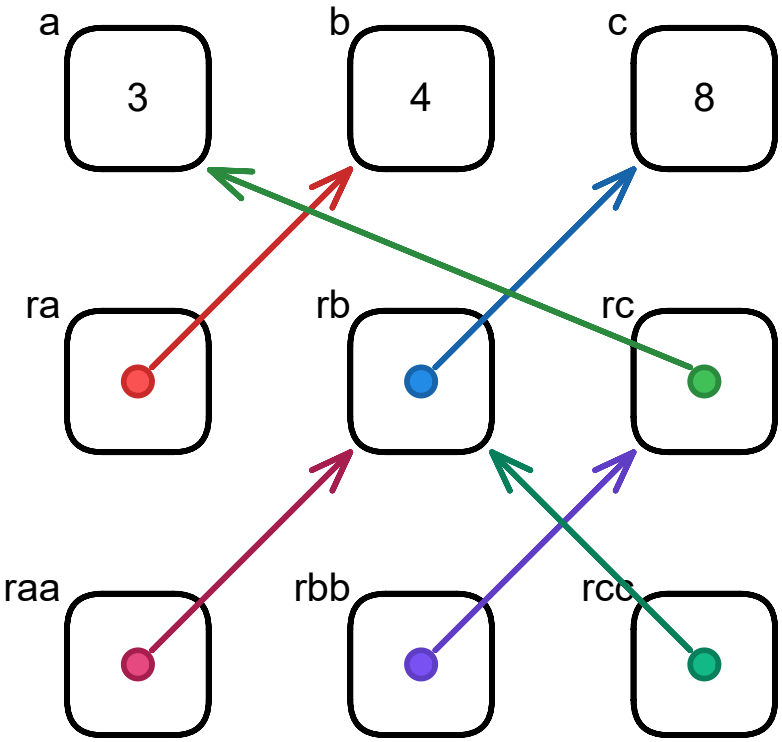
1 Problem - Pointers

Given the following correct program in C,

1. give the correct type definitions for pointer variables `ra`, `rb`, `rc`, `raa`, `rbb`, and `rcc`.

```c
int main() {
        int a, b, c;
        int* ra; int* rb; int* rc; int** raa; int** rbb; int** rcc;
        a = 3; b = 2; c = 1;
        ra = &a;
        rb = &b;
        rc = &c;
        ra = rb;
        raa = &rb;
        rc = *raa;
        rcc = raa;
        rc = &a;
        rbb = &rc;
        rb = &c;
        *ra = 4;
        *rb = *ra + 4;
        /* (*) */
        printf("%d %d %d\n",a,b,c);
        printf("%d %d\n",*ra,*rb);
        printf("%d %d %d\n",**raa,**rbb,**rcc);
}
```

2. draw a picture that shows all of the variables and their contents similar to the picture as shown, for example, in lecture 12, page 12. Also indicate whether the object lives on the stack or on the heap. Your picture should show the variables and their values just before the first print statement (*).



All variables are stored on the stack.

3. show the output from this program.

```
3 4 8
4 8
8 3 8
```

4. write a statement involving a pointer expression using the variables in this program which is ILLEGAL given your declared types.

2 Problem - Compiler Optimization and Aliasing

Assume the following program fragement without any control flow branches (straight line code). Your job is to implement a compiler optimization called “constant folding” for straight line code. This optimization identifies program variables with values that are known at compile time. Expressions that consist of only such variables can be evaluated at compile time. In our project, we do constant folding for ILOC instructions, not on the source code itself.

```
begin
        int a, b, c;
        ... /* some other declarations */
```

```
        a = 4;
        b = 3;
        ... /* no statements that mention ‘‘a’’ or ‘‘b’’ */
        c = a - b; /* c == 1 ? */
        print c;
end.
```

Would it always be safe for the compiler optimization of constant folding to replace the assignment “c = a - b” by “c = 1” ? Note that there are no assignments to variables a or b between “b = 3” and “c = a - b”. The control flow is linear, so there are no branches. Give an example where constant propagation would be not be safe (incorrect) in this situation, without violating any of the above assumptions about the code fragment. Note: You can add declarations of other variables and other statements that do not mention `a` or `b`.

It would not always be safe for the compiler optimization of constant folding to replace the assignment `c = a - b` with `c = 1`. For example, accessing and replacing the value of a or b using a pointer to a, `int *pa = &a` and then setting `*pa = &a` would make it unsafe to apply constant folding because the address of `a` could be anything.

3 Problem — Lexical/Dynamic Scoping

Assume variable names written as capital letters use dynamic scoping and variable names written as lower case letters use static (lexical) scoping. Assume that procedures return when execution reaches their last statement. Assume that all procedure names are resolved using static (lexical) scoping. Show the output of the entire program execution. Label the output with the location of the print statement (e.g.: (*2*): ...)).

```
program main(){
        int A, b;
        procedure f() {
                int c;
                procedure g(){
                        int c;
                        c = 33;
                        ... = ...b...
                        print A,b,c; // 1,2,33 <<<------ (*1*)
                        end g;
                }
```

```
                print A,b; // 4,9 <<<------------ (*2*)
                A = 1; b = 2; c = 3;
                call g();
                print c; // 3 <<<------------ (*3*)
                end f;
        }

        procedure g(){
                int A,b;
                A = 4; b = 9;
                call f();
                print A,b; // 1, 9 <<<----------(*4*)
                end g;
        }

        A = 5; b = 3;
        print A,b; // 5, 3<<<------------ (*5*)
        call g();
        print A,b; // 1, 3 <<<----------(*6*)
        end main;

}
```

N	Output
1	1, 2, 33
2	4, 9
3	3
4	1, 9
5	5, 3
6	1, 3

4 Problem – Lexical Scoping Code Generation

Assume that all variables are lexically scoped.

```
program main(){
        int a, b;
        procedure f(){
                int c;
```

```
                procedure g(){
                        ... = b + c //<<<-------- (*A*)
                        print a,b,c;
                        end g;
                }
                a = 0; c = 1;
                ... = b + c //<<<-------- (*B*)
                call g();
                print c;
                end f;
        }
        procedure g(){
                int a,b;
                a = 3; b = 7;
                call f();
                print a,b;
                end g;
        }
        a = 2; b = 3;
        print a,b;
        call g();
        print a,b;
        end main;
}
```

1. Show the runtime stack with its stack frames, access and control links, and local variables when the execution reaches program point (*A*).
2. Give the ILOC RISC code for the expressions at program points (*A*) and (*B*). The value of the expressions need to be loaded into a register. The particular register numbers are not important here.

5 Problem – Parameter Passing

Assume that you don't know what particular parameter passing style a programming language is using. In order to find out, you are asked to write a short test program that will print a different output depending on whether a call-by-value, call-by-reference, or call-by-value-result parameter passing style is used. Your test program must have the following form:

```
program main(){
        x integer;
        procedure bar(integer a){
```

```
                // statement body of foo
        }
        // statement body of main
        x = 1;
        call bar(x);
        print x;
}
```

The body of procedure bar must only contain assignment statements. For instance, you are not allowed to add any new variable declarations.

1. Write the body of procedure bar such that print x in the main program will print different values for the different parameter passing styles.

```
program main(){
        x integer;
        procedure bar(integer a){
                a = a+1;
                a = x+a;
        }
        x = 1;
        call bar(x);
        print x;
}
```

2. Give the output of your test program and explain why your solution works.

	Call-by-Value	Call-by-Reference	Call-by-Value-Result
x	1	4	3

In a call-by-value parameter passing style, any modifications to `a` do not have any effect on `x`. Therefore, `x` remains as 1 because we never set `x`.

In a call-by-reference parameter passing style, any modifications to `a` have a direct effect on `x` because `a` is a reference to `x`. When we perform addition on `a`, we are also performing addition on `x`. Consequently, `a = a+1` yields `a=2` and `x=2`. The following instruction, `a = x+a`, could be interpreted as `a = 2+2` because `a=x`, yielding the result `x=4`.

In a call-by-value-result parameter passing style, `x`'s value is assigned to the value of `a` only after the function has finished. Therefore, when we do `a=a+1`, we have

that `a=2` but `x=1`. When we add `x+a` in call-by-value-result, we get `x=3` instead, because in the scope of the function bar, `a=2` and `x=1`.