Recursive Square Computation

- Input : k
- Output : square(k) = k^2
- Math : $(k-1)^2 = k^2 - 2k + 1 \Rightarrow k^2 = (k-1)^2 + (2k-1)$
  - $1^2 = 1$
  - $2^2 = 1 + 3$
  - $3^2 = 1 + 3 + 5$
  - ……
  - $k^2 = 1 + 3 + 5 + \ldots + (2k-1)$
- Recursion
  - square(0) = 0
  - square(k) = square(k-1) + (2k-1) for k = {1,2,3,…}
- Next : High level code

Recursive Square Computation : C Program

- Program

```
#include <stdio.h>
int square(int k) {
        if (k==0) return 0;
        else return ( square(k-1) + 2*k-1 );
}
main() {
        int k;
        printf("Please type in a +ve number between 1 and 100: ");
        scanf("%d",&k);
        printf("The input number is %d\n",k);
        printf("The square is %d\n",square(k));
}
```

- Next : See corresponding assembly program. (square.s)

Running square.s on SPIM

- High level program

```
hopi mithuna> gcc sq.c
hopi mithuna> a.out
Please type in a +ve number between 1 and 100: 23
The input number is 23
The square is 529
```

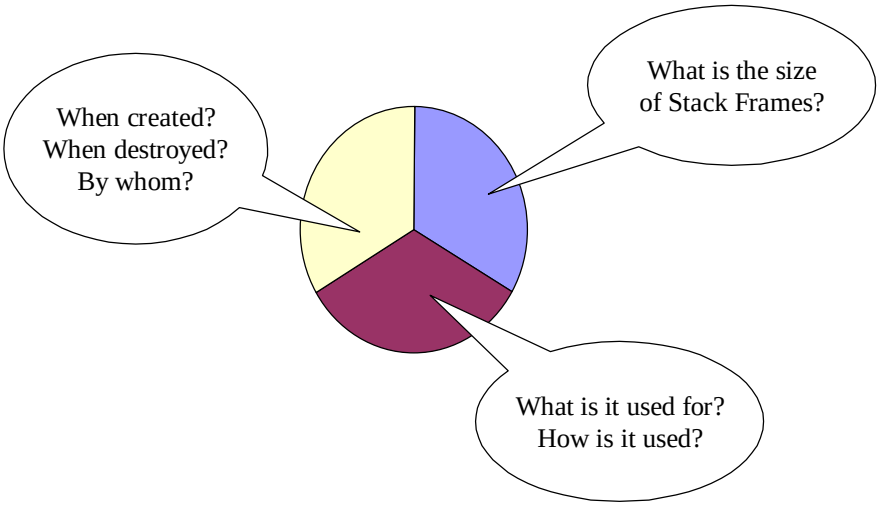
- Assembly program

```
(spim) load "square.s"
(spim) load "spimutils/utils.s"
(spim) run
Please type in a +ve number between 1 and 100:  23
The input number is 23
The square is 529
```

- It works : why does it work?
  - Understanding Stack frames (next)
  - Revisit assembly code and discuss

Understanding Recursion

- Important : Understanding Stack Frames



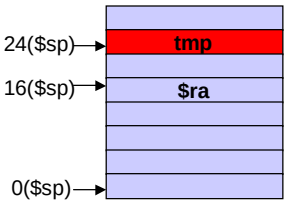
Question # 1

- When created? When discarded? By whom?
  - A new frame for every dynamic invocation of a function
  - Created on entry into function
    - subu $sp, $sp, 32 # First instruction of every function (32 may vary)
  - Deleted just before returning from function
    - addu $sp, $sp, 32 # Penultimate instruction (before return)
- Exercise : Show frames on stack for the following call sequence.
  - main() calls foo()
  - foo() calls foo() recursively
  - foo() calls foo() recursively
  - foo() calls bar()
  - Bar() executes return
  - foo() executes return
  - foo() calls bar()
  - bar() executes return
  - foo executes return
  - foo() executes return
  - main() executes exit()

Question #2

- What is it used for? How is it used?
- Used to save
  - Variables of local scope and arguments
  - Temporary variables (used in evaluation of long expressions)
  - Registers (callee saved which function modifies, etc)
- Each such variable is assigned a place in the stack frame
  - Addressed relative to Stack pointer (ignore frame pointer for now)
  - E.g. Variable tmp is assigned to location 24($sp) say, of a 32 byte stack frame
  - Assembly code to compute tmp++ is

```
lw $t0, 24($sp)
addi $t0, $t0, 1
sw $t0, 24($sp)
```



Question # 3

- What is the size of Stack Frames?
- Big enough to
  - Hold all arguments and variables of local scope
  - Temporary variables (used in evaluating large expressions)
  - Save callee saved registers and
  - Save caller saved registers we want to retain
- Larger Stack Frame than the minimum required
  - Correctness not affected
  - Performance may be affected
- Conventions
  - At least 24 bytes
  - Double word aligned (divisible by 8)
- Next: Revisit assembly code and discuss