Tail recursion in C
Take this C code:

```c
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
```
Stack

```c
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
```
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
int always_zero(i) {
    if(i==0) return 0;
    return always_zero(i-1);
}

void main() {
    always_zero(5);
}
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
int always_zero(i) 
{
    if (i == 0) 
        return 0;
    return always_zero(i-1);
}

void main() 
{
    always_zero(5);
}
```c
int always_zero(int i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
```
Stack

```c
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
```
```c
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
```
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
void main()
{
    always_zero(5);
}

int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

always_zero(4)
always_zero(5)
main()
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
A lot of unnecessary work and storage! Can we do better?
Tail recursion

```
int always_zero(i) {
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main() {
    always_zero(5);
}
```
Tail recursion

```c
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
```
Tail recursion

```c
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
```
Tail recursion

int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
Tail recursion

```c
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
```
Tail recursion

```c
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
```
Tail recursion

```c
int always_zero(i)
{
    if(i==0)
        return 0;
    return always_zero(i-1);
}

void main()
{
    always_zero(5);
}
```
Tail recursion

- Just discard unnecessary functions from the stack
- Same results
- O(1) space instead of O(n) space
- Faster too!

- Can we do this for all recursive algorithms?
int always_zero(i)
{
    if(i==0)
        return 0;
    return 5*always_zero(i-1);
}

void main()
{
    always_zero(5);
}
Can we do the same trick?

- No! Results returned by `always_zero(0)` need to be further processed by `always_zero(1)`
- Results from `always_zero(0)` cannot be returned directly to `main()`
- Need to pass through `always_zero(1)`, `always_zero(2)`, `always_zero(3)`, ...
Tail recursion only works when we need no further processing.
### Terminology

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Iteration</th>
<th>Recursion</th>
</tr>
</thead>
</table>
| C      | for(i=0; i<100; i++) { } | int f(int i) {
|        |           |     if(i==0) {
|        |           |         return 0;
|        |           |         return f(i-1); |
|        |           | }         |
|        | Tail recursion | Normal recursion |
|        | int f(int i) {  | int f(int i) {
|        |     if(i==0)  |     if(i==0) {
|        |       return 0; |       return 1; |
|        |     else      |     else |
|        |       return i*f(i); |       return i*f(i); |
|        | }            | }         |