# C and MIPS Review

CS61C Summer 2004
Navtej Sadhal

## Dynamic Tree Structure in C

- We wish to develop a tree structure comprised of nodes with a flexible number of children.
- Each node may have a different number of child nodes. Children are the same type as their parent.
- The structure must contain an integer of data, a integer indicating the number of children, and an array of pointers to the children. It should look like this in memory:

| Data | N | ChPtr1 | ChPtr2 | … | ChPtrN |
|---|---|---|---|---|---|

Integers and pointers are both 32 bits each.

## Dynamic Tree Structure in C

How can we define a `struct node` in C that will have this structure in memory?

| Data | N | ChPtr1 | ChPtr2 | … | ChPtrN |
|---|---|---|---|---|---|

## Dynamic Tree Structure in C

How can we define a `struct node` in C that will have this structure in memory?

| Data | N | ChPtr1 | ChPtr2 | … | ChPtrN |
|---|---|---|---|---|---|

```
struct node {
     int data;
     int numChildren;
```

## Dynamic Tree Structure in C

How can we define a `struct node` in C that will have this structure in memory?

| Data | N | ChPtr1 | ChPtr2 | … | ChPtrN |
|---|---|---|---|---|---|

```
struct node {
     int data;
     int numChildren;

     struct node * children[]
}
```

## Dynamic Tree Structure in C

How can we define a `struct node` in C that will have this structure in memory?

| Data | N | ChPtr1 | ChPtr2 | … | ChPtrN |
|---|---|---|---|---|---|

```
struct node {
     int data;
     int numChildren;

     struct node * children[]
}
```

What does this say about the size of our struct?

## Dynamic Tree Structure in C

How can we define a `struct node` in C that will have this structure in memory?

| Data | N | ChPtr1 | ChPtr2 | … | ChPtrN |
|---|---|---|---|---|---|

```
struct node {
     int data;
     int numChildren;

     struct node * children[]
};
```

What does this say about the size of our struct?
It's variable!

## Traversing the tree

- We would now like to write a function which will search the tree for an integer. If it exists, return 1, otherwise return 0.

```
bool contains(int num, struct node * tree) {
```

## Traversing the tree

- We would now like to write a function which will search the tree for an integer. If it exists, return 1, otherwise return 0.

```
bool contains(int num, struct node * tree) {
     int i;
     if (tree == null) return 0;
     if (tree->data == num) return 1;
     for (i=0; i < tree->numChildren; i++)
            if (contains(num, (tree->children)[i]))
                   return 1;
     return 0;
}
```

## Traversing the tree

- We would now like to write a function which will search the tree for an integer. If it exists, return 1, otherwise return 0.

```
bool contains(int num, struct node * tree) {
     int i;
     if (tree == null) return 0;
     if (tree->data == num) return 1;
     for (i=0; i < tree->numChildren; i++)
            if (contains(num, (tree->children)[i]))
                   return 1;
     return 0;
}
```

- Now translate this to MIPS. Assume num is in $a0 and tree is in $a1. Put the result in $v0.

## Translating C to MIPS

```
contains:
        addiu   $sp, $sp, -20
        sw      $ra, 0($sp)
        sw      $s0, 4($sp)
        sw      $s1, 8($sp)
        sw      $s2, 12($sp)
        sw      $s3, 16($sp)

        mv      $s0, $a0
        mv      $s1, $a1

        beqz    $s1, end0
        lw      $t0, 0($s1)
        beq     $t0, $s0, end1

        mv      $s2, $0
        lw      $s3, 4($s1)
        sll     $s3, $s3, 2
loop:
        beq     $s2, $s3, end0

        mv      $a0, $s0
        addu    $a1, $s1, $s2
        lw      $a1, 8($a1)
        jal     contains
        bne     $v0, $0, end1
        addiu   $s2, $s2, 4
        j       loop

end1:
        li      $v0, 1
        j       end

end0:
        mv      $v0, $0

end:
        lw      $ra, 0($sp)
        lw      $s0, 4($sp)
        lw      $s1, 8($sp)
        lw      $s2, 12($sp)
        lw      $s3, 16($sp)
        addiu   $sp, $sp, 20
        jr      $ra
```

## Translating C to MIPS

- Why is this solution inefficient?

## Translating C to MIPS

- Why is this solution inefficient?
  - It does stack operations every time, even when tree is null or num is found in top level node.
    - These cases happen at every leaf of the tree

## Translating C to MIPS

- Why is this solution inefficient?
  - It does stack operations every time, even when tree is null or num is found in top level node.
    - These cases happen at every leaf of the tree
- How can we improve this?

## Translating C to MIPS

- Why is this solution inefficient?
  - It does stack operations every time, even when tree is null or num is found in top level node.
    - These cases happen at every leaf of the tree
- How can we improve this?
  - Do these cases before we do stack operations because $sX registers are unnecessary for these cases anyway.

## Using Malloc

- Now write a function in C which will add a node as a child to another node:

```
void addChild(struct node ** parent, struct node * child) {
```

## Using Malloc

- Now write a function in C which will add a node as a child to another node:

```
void addChild(struct node ** parent, struct node * child) {
     int i, numChildren = (*parent)->numChildren;
     struct node * temp = malloc(2*sizeof(int) +
            (numChildren + 1)*sizeof(struct node *));
     temp->data = (*parent)->data;
     temp->numChildren = numChildren + 1;
     for (i=0; i<numChildren; i++)
            (temp->children)[i] = ((*parent)->children)[i];
     (temp->children)[numChildren] = child;
     *parent = temp;
}
```