CS 61C
Spring 2010

More C & Memory Management

Scott Beamer (cs61c-ta)

Week 3 (2/2)

Quick Array Problem

Fill in the function to make it compute the dot product of a•b

```
int dotProduct(int a[], int b[], int length) {
      int i, dotP = 0;
      for(i = 0; i < length; i++) {
            dotP += a[i] * b[i];
      }
      return dotP;
}
```

Dynamic Memory Allocation Summary

- `int sizeof(`*`datatype`*`)` - returns the number of bytes needed to hold *`datatype`*
- `void* malloc(int numBytes)` - returns address of dynamically allocated block that is `numBytes` long, or returns `0` if it can't satisfy that request
- `void free(void *ptr)` - releases the memory that `ptr` points to

Summary of struct

- Composes simpler data types to make data structures
- Can get an element by: *structInstanceName.elementName*
- If passed by a pointer, ptrName->elementName instead of (*ptrName).elementName

Summary of typedef

- `typedef replaceWith  searchFor;`
- For declarations, replaces `searchFor` with `replaceWith`

Linked List Example

```
typedef char *String;
typedef struct Node {
      String value;
      struct Node *next;
} NodeStruct;
typedef NodeStruct *List;

List cons (String s, List list) {
      List node = (List) malloc(sizeof(NodeStruct));
      node->value = (String) malloc (strlen(s) + 1);
      strcpy(node->value, s);
      node->next = list;
      return node;
}
```

Summary of union

- Used to make more general data types (syntax is like struct)
- Only 1 type is valid at a given time and it is programmer's responsibility to know which
- Often another variable is used to hold which type is there

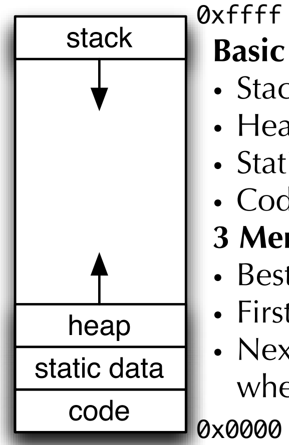
```
  union Number {              // let numType hold realNum's type
       float fVal;            if(numType == FLOAT)
       double dVal;                 realNum.fVal = 3.14f;
} realNum;                    else if(numType == DOUBLE)
                                    realNum.dVal = 3.14;
```

General Linked List Problem

Change the declaration from the Linked List Example to handle int's in addition to Strings by using unions. Make a function that sums the values of the elements assuming they are ints.

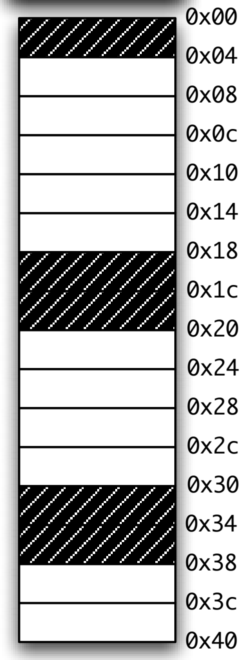
```
typedef char *String;
typedef struct Node {
      union {
            int intValue;
            String strValue;
      } value;
      int valueType;
      struct Node *next;
} NodeStruct;
typedef NodeStruct *List;

int sumList(List list) {
      if(list == NULL)
            return 0;
      return list->value.intValue + sumList(list->next);
}
```



Basic C Memory Management (4 segments)
- Stack - grows down - holds local variables
- Heap - grows up - where malloc() requests space
- Static Data - fixed size - holds global variables
- Code - fixed size - immutable - where instructions for program are

3 Memory Allocation Schemes
- Best-fit - choose the smallest block that satisfies the request
- First-fit - choose the first block that satisfies the request starting from the front
- Next-fit - choose the first block that satisfies the request starting from the where the last request finished



Memory Allocation Problem

Fill in the table, listing the starting address that each request will be satisfied by. Assume that:
- The diagram on the left is the initial conditions
- Next-fit will start originally from the beginning
- All schemes will choose the lowest address in the selected range

Request	Best-fit	First-fit	Next-fit
4 bytes	0x38	0x04	0x04
4 bytes	0x3c	0x08	0x08
16 bytes	0x20	0x20	0x20
8 bytes	0x04	0x0c	0x38
12 bytes	0x0c	can't	0x0c