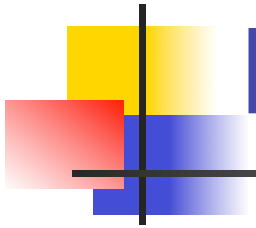
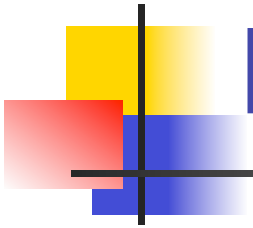
CSCI 143 Software Eng I

Rhys Price Jones



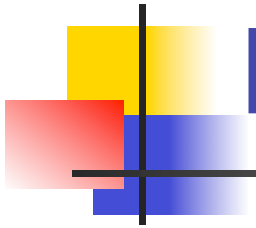
Files

- stdin scanf(..)
- stdout printf(..)
- stderr fprintf(stderr,...)
- Redirection < >
- Look at first exercises



File I/O

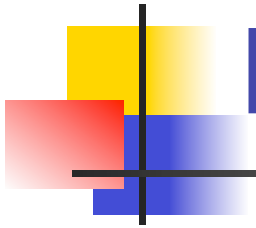
- FILE *fileptr;
  fileptr = fopen(”filename.ext”, “w”);
  fprintf(fileptr, “%d\n”, whatever);
  fscanf(fileptr, “%d”, &whatever);
  fclose(fileptr);
  - “r” for read,
  - “w” for write,
  - “a” for append
  - other modes b-binary, r+, w+, a+
  - combined modes rb
- fgetc(fileptr), fputc(fileptr,c)



Functions

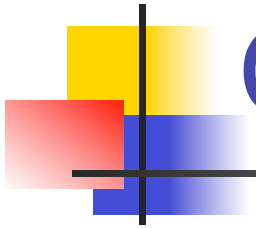
- Definition must appear before use
  - but you can use a function prototype
    - double comingsoon(int, int)
    - and recursion is ok

```
#include <stdio.h>
double comingsoon(double, int);
main(int argc, char *argv[]) {
  printf("argv1 is %s\n", argv[1]);
  int x = atoi(argv[1]);
  int i = atoi(argv[2]);
  printf("value of %d^%d is %f\n", x, i, comingsoon(x,i));
}
double comingsoon(double x, int n) { // power function
  return (n==0) ? 1.0 : x*comingsoon(x,n-1);
}
```



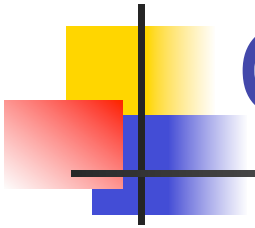
Parameters

- call by value:
  - caller places value of x on stack
  - function uses (or abuses!) that copy
  - on return, stack frame disappears
- in true call-by-value
  - callees cannot modify caller's data
  - Always so in C?
- Try the inclass exercise



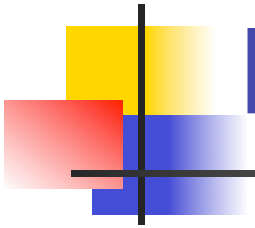
Call by reference

- caller places the address of the parameter on the stack
- callee can access and update the contents by dereferencing the pointer
- The caller's pointer is not modified
- But its contents may be
- Next exercise



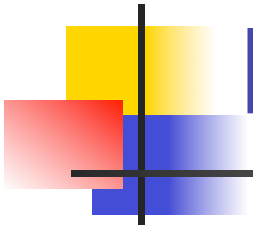
Global local static parameter

- global: declared outside of all functions
- static: declared as static inside functions (maintain value between calls)
- local: declared inside functions (do not maintain values between calls)
- parameter: for passing values



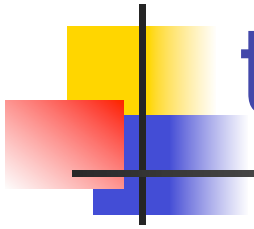
Math library

- see in-class exercises
- ceil(x)
- floor(x)
- fabs(x)
- etc.



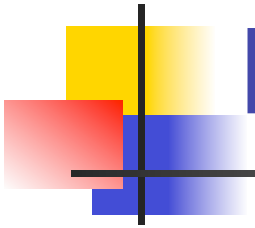
Enumerated types

- enum days {mon, tue, wed, thur, fri}
- implemented with ints (0,1,2,3,4)
- declarations via
  - enum days today;



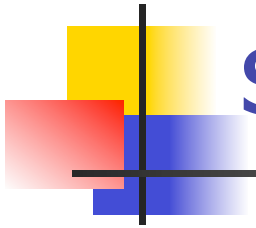
typedef

- will use in lab 3
- creates declaration shortcuts
- typedef enum days {mon, tue, wed, thur, fri} dayname
  - allows you to declare
    - dayname today;
    - today = wed;



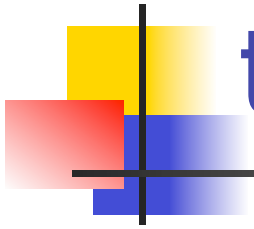
Notes on typedef

- syntactic shortcut, compiler does replacement in pre-compilation pass
- the type's name appears in the "variable" position:
  - typedef float *fpointer
- mostly useful for naming structs



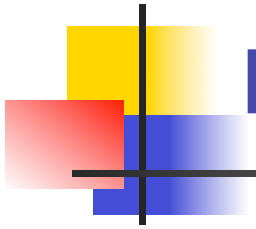
structs

- group together data fields
  - objects in Java also group data fields, but objects also allow methods
- structvar.fieldname accesses a particular field
- structptr->fieldname does the same if we have a pointer to the struct
- same as (*structptr).fieldname



typedefs and structs

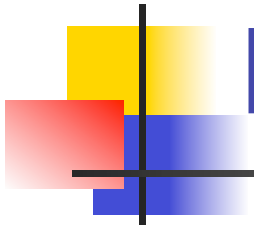
- go together well
- and can make for elegant code
  - even in C! :-)
- especially good for nodes in dynamic linked structures
- We'll meet linked lists and binary search trees in Lab 3.



malloc()

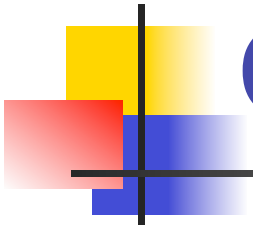
- is in <stdlib.h>
- void *malloc(size_t size)
  - What's with void *?
  - What's size_t?
- either #include <stdlib.h>
- or declare [MyType] *malloc(int);

The *malloc*() function shall allocate unused space for an object whose size in bytes is specified by *size* and whose value is unspecified. If the space cannot be allocated, a null pointer shall be returned.



Reading words

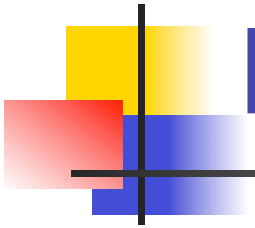
- and storing them in a buffer
- with the ability to unget a word
- code in in-class exercise



Comparing words

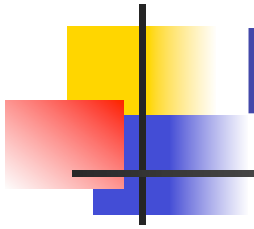
```
int strcomp(char *s, char *t)
{
   for ( ; *s == *t++; s++ )
      if (!*s) return 0;
   return (*s - *t);
}
```

- returns neg, 0, or pos
- according as ...?



Lists

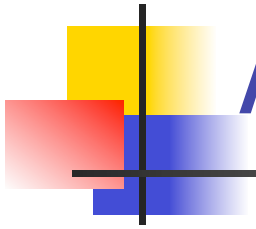
- of words
- ```
  typedef struct listnode {
    char *word;
    struct listnode *next;
  } List, *ListP;
  ```
- A list is
  - either empty (NULL)
  - or a listnode followed by a list
- We'll use ListP more that List
  ```



Building Lists

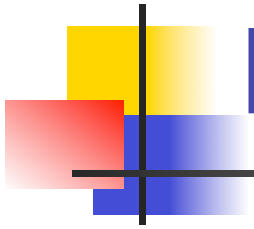
- `construct()` adds a new item to the head of an existing list
- process bootstraps with `NULL`

```
ListP construct(char *data, ListP n) {
  ListP new = malloc(sizeof(List));
  new->word = data;
  new->next = n;
  return new;
}
```



Accessing

- first element
  - ```
    char *head(ListP lp) {
      return lp->word;
    }
    ```
- rest of list
  - ```
    ListP rest(ListP lp) {
      return lp->next;
    }
    ```



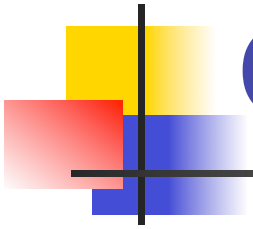
Finding length of list

```
int length(ListP lp) {
   return (lp == NULL) ?
          0 :
          1 + length(rest(lp));
}
```

Testing membership in list

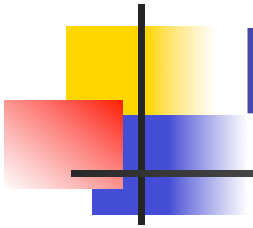
```
isMember(char *w, ListP lp) {
  if (lp == NULL) return 0;
  if (strcomp(w, lp->word)==0)return 1;
  return isMember(w, rest(lp));
}
```

- returns 1 for true
  - 0 for false



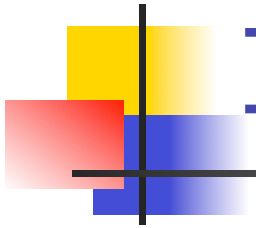
Constructing from the back

- ```
  ListP addLast(char *w, ListP
  lp) {
    return (lp == NULL) ?
      construct(w, lp) :
      construct (head(lp),
  addLast(w, rest(lp)));
  }
  ```
  ```



More list functions

- printing a list?
- full program in in-class exercises
- removing a member?
- doubling members?

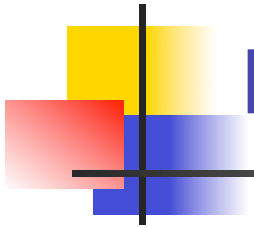


Immutable lists

- is what we've done so far
- disadvantages
  - create lots of garbage
  - need to free malloc'ed memory
  - better for languages like Java or Scheme with automatic garbage collection
- advantages
  - easy and relatively error-free

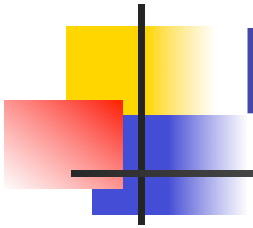
Using immutable lists

- Typical assignment:
  - l = modify(l)
- If modification creates garbage, be ready to free it
  - or else???
- Reversing a list
  - ```
    ListP reverse(ListP lp) {
      return (lp == NULL) ?
        lp :
        addLast(head(lp), reverse(rest(lp)));
    }
    ```
    ```



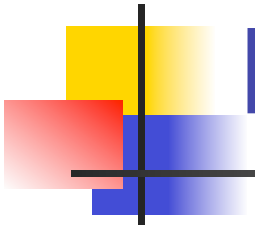
reverse creates garbage

- how much?
- so what?
- let's measure it
- suggested experiment?



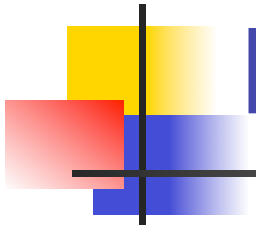
Mutable lists

- use assignment in the structs
- re-use existing nodes
- pro:
  - avoids garbage collection
- con:
  - very frequent cause of error



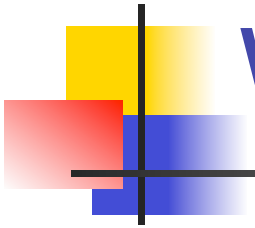
Memory Management

- in C: programmer's responsibility
  - dangers include memory leaks (failure to free()) and destruction of data (failure to malloc() when needed)
- in Java: programmer responsible for malloc() [but it's called "new"]
- garbage collection is automated
  - runs in its own thread
- Speaking of Java...



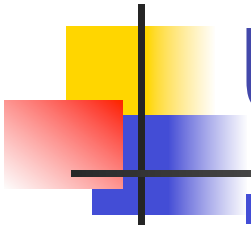
Binary Search Trees

- In lab3 you'll work with a linked list of ints and a BST for storing and accessing words
- In this lecture you've worked with a linked list of words
- So now let's develop a BST for ints



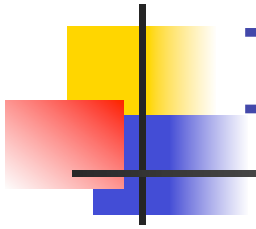
What is a BST?

- either it is empty (NULL)
- or it has data (an int) and two children (left and right) both of which are BSTs
- ```
  typedef struct BSTnode {
     int data;
     struct BSTnode left;
     struct BSTnode right;
  } BST, *BSTptr;
  ```
  ```



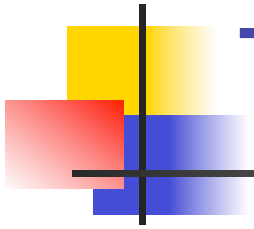
Using a BST

- We'll write a program to read and store all the ints in a file, count the occurrences of each and print them in order with their counts.
- Modify the typedef:
- ```
  typedef struct BSTnode {
     int data; int count;
     struct BSTnode left;
     struct BSTnode right;
  } BST, *BSTptr;
  ```
  ```



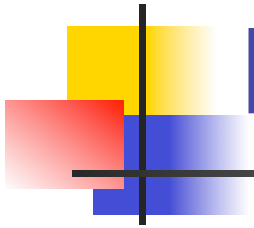
Insertion into a BST

```
BSTptr insert(BSTptr t, int x) {
  if (t == NULL) {     /* insert into empty tree */
    t = talloc();
    t->data = x;
    t->count = 1;
    t->left = t->right = NULL;
  }
  else if (x == t->data)
    t->count++;
  else if (x < t->data) t->left = insert(t->left, x);
  else t->right = insert(t->right, x);
  return t;
}
```



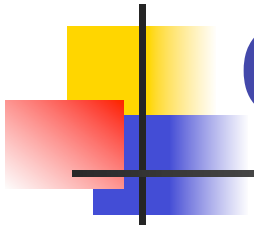
Traversal of a BST

```
void print_tree(t)
  BSTptr t;
{
  if (t == NULL) ; /* do nothing */
  else {
    print_tree(t->left);
    printf("%d appears %d times\n",
            t->data, t->count);
    print_tree(t->right);
  }
}
```



Reading ints

```
int done = 0;
int digit(char ch) {
  return  (ch>='0'&&ch<='9');
}
int getint() {
  if (done) return (int) EOF;
  char *ch;
  char buffer[20];
  ch = buffer;
  while (!digit(*ch = fgetc(inFile)) &&
         (*ch != EOF));
  if (*ch == EOF) return (int) EOF;
  while (digit(*++ch = (fgetc(inFile))));
  if (*ch == EOF) done = 1;
  *ch++ = '\0';
  return atoi(buffer);
}
```



On to Java...