terminal vars
code output

Functions make life easier

Basics of Functions

return-type (default to int) function-name (argument declarations) {
declarations & statements }

program: set of def^n's of vars & fn's

functions communicate thru arguments & values from other fn's & external variables
return is optional

Functions Returning Non-integers

return-type must be correct
good to explicitly state it when calling fn

w/o stating explicitly, if fn is defined & compiled in a different file, main will interpret as int

pay attention to potentially losing info ex: double → int

External Variables

C program has external objects (variables & fn's)
↳ "internal" means args & vars inside a fn

these external vars can be used in many fn's

functions are always external → can't be in another fn

good to communicate between fn's

using same data for 2 vars? → external var!

ungetch remembers char put back on input, puts pushed-back chars into shared buffer

getch delivers next input char. to be considered, gets from buffer, or getchar if empty buffer
also need to think of buffer's index

Scope Rules

all fn's & external vars making a C program don't need to be compiled @ same time

Keep in mind:
- how are declarations written so vars properly compiled?
- " arranged so all pieces properly connected when loaded?
- " organized so there are only 1 copy?
- " external vars initialized?

Scope of a name is part of program within which name can be used
var can only be used within fn, unless external

external var's scope is from declaration to End Of File

declaration - announce properties of var (type)

definition - " " & set aside storage

can only be 1 defn of an external var among all files making up program

Header Files

best to put main fn in a main.c file

want to centralize shared defn's & declarations
↳ header file calc.h

#include "calc.h"

tradeoff between file having only info needed & working w/many files

w/larger projects, good to use multiple header files

Static Variables

make a variable or fn private

static type var_name ;

limits scope of object to rest of source code being compiled

Register Variables

advises compiler a variable will be heavily used, only for automatic vars

in theory, may result in smaller, quicker programs

```
register type var_name;
```

Block Structure

declarations of vars can be inside any block, not just a fn

scope stuff

Initialization

for external & static vars, initializer must be a constant expression
→ initialization done once, before program begins execution

for automatic & register vars, initializer is not restricted to being constant
→ done each time the fn or block is entered

initializations of automatic vars are shorthand for assignment

initializing a char array uses quotes, can also use braces

```
char pattern[] = "ould"
char pattern[] = {'o', 'u', 'l', 'd'}
```

Recursion

recursive fn's - fn calling itself directly or indirectly

each invocation gets fresh set of automatic vars

good for sorting algorithm

not very storage & fast, but very compact