Notes

- Syllabus is now available on Blackboard
- A Discussion Forum has been created for informal interactions
  - Treat this as a study group
  - Use it to clarify questions as well as to test your knowledge
  - Ask frequently - reply even more frequently if you can
- Another interesting read
  - "The C Programming Language" (Second Edition) by Brian Kernighan and Dennis Ritchie.
- Your program must compile and work correctly on the machines provided by the Information Technology Services (ITS)
  - You can log on to these machines remotely.
  - You will need a client like Putty, your NetID, and your Password
  - Connect to itsunix.albany.edu
Systems Programming

- The programmer will make assumptions about the hardware and other properties of the system that the program runs on, and will often exploit those properties, for example by using an algorithm that is known to be efficient when used with specific hardware.
- Usually a low-level programming language or programming language dialect is used that:
  - can operate in resource-constrained environments
  - is very efficient and has little runtime overhead
  - has a small runtime library, or none at all
  - allows for direct and "raw" control over memory access and control flow
  - lets the programmer write parts of the program directly in assembly language

- The use of automatic garbage collection is not common.
- Debugging is hard to do.
System Programmer
System Programmer

- Great C programmer
  - Usually has to manage memory directly
- Deals with concepts extremely close to the hardware level
  - Registers and memory locations
  - Even fiddling with the individual bits of a small word of memory
- Manipulates files and directories
- Uses any standard system or library calls
  - Checks the return value from each function call for errors
A Single File Program

File: prog.c

```c
#include <stdio.h>
int main(void) {
    int max(int, int); float average(int, int);
    int i = 17, j = -20;
    printf("%d\n", max(i, j)); printf("%f\n", average(i, j));
    return 0;
}
int max (int a, int b) {
    if (a >= b)
        return a;
    else return b;
}
float average (int a, int b) {
    return (a+b)/2.0;
}
```
A Single File Program

- Unix command to generate a.out:
  - gcc prog.c

- Difficulties
  - If the source file is large, it is harder to fix the syntax errors.
  - Even if all errors are in just one function, the whole file must be recompiled.

Solution: Split program into several small files.
Program Split in Multiple Files

► File: prog.c

```c
#include <stdio.h>

int main(void) {
    int max(int, int); float average(int, int);
    int i = 17, j = -20;
    printf("%d\n", max(i, j)); printf("%f\n", average(i, j));
    return 0;
}

int max (int a, int b) {
    if (a >= b)
        return a;
    else return b;
}

float average (int a, int b) {
    return (a+b)/2.0;
}
```

► File: main.c

```c
#include <stdio.h>

int main(void) {
    int max(int, int); /* Returns the larger of the two inputs. */
    float average(int, int); /* Returns the average of the two inputs. */
    int i = 17, j = -20;
    printf("%d\n", max(i, j));
    printf("%f\n", average(i, j)); return 0;
}
```

► File: max.c

```c
int max (int a, int b) {
    if (a >= b)
        return a;
    else return b;
}
```

► File: avg.c

```c
float average (int a, int b) {
    return (a+b)/2.0;
}
```
Program Split in Multiple Files

- Unix command to generate a.out:
  - gcc main.c max.c avg.c

- Better method
  - First compile the files separately
    - gcc -c main.c
    - gcc -c max.c
    - gcc -c avg.c
  - and then link them to get the executable version.
    - gcc main.o max.o avg.o

- Remarks
  - The “-c” option specifies “compile only”.
  - Note the use of gcc for linking the object files.
  - Additional information is given in Handout 1.1.
A More Complicated Example

File: main.c

```c
#include <stdio.h>
int x, y;
float z[10];  /* x,y,z : global. */

int main(void) {
    void xy_change(int);
    void z_change(int, float);

    x = 15; y = 17; z[3] = 5.2;
    xy_change(3); printf("%d %d\n", x, y);
    z_change(3, 7.4); printf("%f\n", z[3]);
    return 0;
}
```

File: funct.c

```c
/* x, y, z : externally defined. */
/* Size not specified for z. */

extern int x, y;
extern float z[];

void xy_change (int a) {
    x += a; y -= a;
}

void z_change (int a, float x) {
    z[a] = x;
}
```
Example with Header Files

File: main.c

```c
#include <stdio.h>
#include "globals.h"

int main(void) {
    void xy_change(int);
    void z_change(int, float);

    x = 15;  y = 17;  z[3] = 5.2;

    xy_change(3);  printf("%d %d\n", x, y);
    z_change(3, 7.4);  printf("%f\n", z[3]);

    return 0;
}
```

File: funct.c

```c
#include "externs.h"

void xy_change (int a) {
    x += a;  y -= a;
}
void z_change (int a, float x) {
    z[a] = x;
    x += a;  y -= a;
}
void z_change (int a, float x) {
    z[a] = x;
}
```

File: globals.h

```c
int x, y;
float z[10];
```

File: externs.h

```c
extern int x, y;
extern float z[];
```
Example with Header Files

- Unix command to generate a.out:
  
  ```bash
  gcc -c main.c
  gcc -c funct.c
  gcc main.o funct.o
  ```

- Remarks
  
  - Header files are not specified in the compile command.
What else can be Included in Header Files

- Symbolic Constants
  ```c
  #define MINKEY 1
  #define MAXKEY 100
  ```

- Structure Definitions
  ```c
  struct key_record {
    int value;  struct key_record *next;
  };
  typedef struct key_record* keyptr;
  ```

- Function prototypes
  ```c
  void insert_key(int);
  void print_list(void);
  ```

- A complete example is provided in Handout 1.2.
Summary Regarding Header Files

- constants.h
- struct_def.h
- globals.h typically included in the source file containing `main`.
- externs.h typically included in source files containing functions that access the global variables.
- prototypes.h