CS1003: Intro to CS, Summer 2008

Lab #03

Instructor: Arezu Moghadam
arezu@cs.columbia.edu

Recap...

- Math operators
- Arrays (assignment and reference)
- Strings
  - string manipulation
  - fgets
  - sscanf
Function prototypes

- Usually, you declare variables before you can use them
  - similar with functions
  - however, you can
    - declare a function prototype at the beginning of the program
    - define the actual function workings later on

- Example
  - int add (int a, int b);

- This will be important in HW2

---

Function prototypes – code

```c
#include <stdio.h>

int add (int a, int b);

int main() {
  int c;
  c=add(2, 3);
  printf("The total of 2 and 3 is %d\n", c);
}

int add (int first_number, int second_number) {
  int total;
  total = first_number + second_number;
  return total;
}
```
Function prototypes – code II

```c
#include <stdio.h>

int add (int a, int b);

int main(int argc, char *argv[]) {
    int c, x, y;
    x=atoi(argv[1]);
    y=atoi(argv[2]);
    c=add(x, y);
    printf("The total of %d and %d is %d\n", x, y, c);
}

int add (int first_number, int second_number) {
    int total;
    total = first_number + second_number;
    return total;
}
```

More examples:

```c
#include <stdio.h>

// defining all my function prototypes
int add (int a, int b);
int minus (int a, int b);
int mult (int a, int b);
float div (int a, int b);

int main(int argc, char *argv[]) {
    int addanswer, minusanswer, multanswer, x, y;
    float divanswer;
    // reading in all the input
    x=atoi(argv[1]);
    y=atoi(argv[2]);
    // performing calculations and printing the result
    addanswer=add(x, y);
    minusanswer=minus(x, y);
    multanswer=mult(x, y);
    divanswer=div(x, y);
    printf("The respective calculations of %d and %d are %d, %d, %d and %f\n", x, y, addanswer,
            minusanswer, multanswer, divanswer);
}
```
Example contd...

```c
// The add function
int add (int first_number, int second_number) {
    int total;
    total = first_number + second_number;
    return total;
}

// The subtraction function
int minus (int first_number, int second_number) {
    int total;
    total = first_number - second_number;
    return total;
}

// The multiplication function
int mult (int first_number, int second_number) {
    int total;
    total = first_number * second_number;
    return total;
}

// The division function – note that this one returns a float
float div (int first_number, int second_number) {
    float total;
    total = (float) first_number / (float) second_number;
    return total;
}
```

BTW (a couple of comments about comments and style)

- Use comments
- Use tabs to write code cleanly
- Identify yourself as the author
Conditionals

- Conditional statements
  - if
  - switch

- if
  - need to know <, >, ==, !=
  - usage:
    ```
    if (expr) {stmt...}
    else if (expr) {stmt...}
    else {stmt}
    ```

  - when do you not need {}?
  - if followed by another if
    - if (something) do something;
    - if (something else) do something else;
  - The default case is the final else

- Correctness
  - if (strcmp(string1, string2)) do something?
  - if (strcmp(string1, string2)==0) do something?
Conditionals II

- Switch

```
switch (val) {
    case 1:
        do some work;
        break;
    case 2:
        do some work; // you don't have to necessarily have
        // stuff here
        break;
    case 3:
        do some work;
        break;
    default: // if needed
        do some work;
        break;
}
```

- What is the break statement?
- What happens if you don’t use break? Why? (answer using the following example)

---

Switch-case example:

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    int a, b;
    char op;
    char line[100];
    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d %c %d", &a, &op, &b);
    printf("Integer value of your requested operation: %d\n", op);
    switch(op){
        case '+':
            printf("%d + %d = %d\n", a, b, (a+b));
            break;
        case '*':
            printf("%d * %d = %d\n", a, b, (a*b));
            break;
        case '-':
            printf("%d - %d = %d\n", a, b, (a-b));
            break;
        case '%':
            printf("%d mod %d = %d\n", a, b, (a%b));
            break;
        case '/':
            printf("%d / %d = %d\n", a, b, (a/b));
            break;
        default:
            printf("Operation is not supported!!\n");
    }
    return 0;
}
```
Goto and the evils of it...

- DON'T USE GOTO
- What is GOTO
- Why is it a problem?

---

Loops

- Iteration/loops
  - While
  - For
  - Do while
- Difference between conditionals and loops

---
Loops II

- While
  - usage:
    - `while (cond) {stmt...}`
  - break;
  - continue;
- code
  ```
  while(current_number<100) {
    do something; // what is wrong
    i++; // or i- as the case may be
  }
  ```
Loops III

Do while

usage:

```plaintext
do {
    blah;
} while (i>0);
```

Again, remember that the value of ‘i’ needs to be changed

Loops IV

For

usage:

```plaintext
for ( ... ; ... ; ... ) {
    do something here;
}
```

There is other acceptable syntax (sort of)

BTW, this is where the ++i and i++ becomes relevant and useful

Everything in for can be done in a while

Think about it
Loops V

- The comma operator
  - Things are evaluated from left to right
- for (sum=0, i=1; i<=n; ++i)
  \[ \text{sum} += \text{i}; \]
- for (sum=0, i=1; i<=n; \text{sum} += \text{i}, ++\text{i})
  \[ ; \]
- for (sum=0, i=1; i<=n; ++i, \text{sum} += \text{i})
  \[ ; \] // this may give wrong results as i is
  \[ // \text{incremented before added to sum} \]

Loops VI

- Why can we use the ; just like that
- Infinite loops – beware
  - while (1) { ...}
  - for ( ; ; ) {...}
    - Use it at your own risk
    - Use it instead of running your program again and again
What does the following do?

```c
for (i = 1; i <= 10; ++i )
   ;
sum += i;
```

For loop example

- Write a program that accepts x (a float) and n (an integer) from user and raises x to the power n and prints out the result.
While loop example

```c
#include <stdio.h>

int main(void){
    char c;

    printf("Start entering your characters...\n");
    do {
        c = getchar();
    } while(c != 'E');

}
```

Assignment

- Read Ch. 8 and 9 from the Practical C Programming book
- Start reading Ch. 7

- HW2