GLG 490 Numerical Methods
Intro to C Programming – part 5

Reading variables from the keyboard

Often, it is useful to make your program interactive with the user. You can read input in from the keyboard using the scanf function. Note, this is similar to the sscanf function, except that a file pointer is not required for scanf.

Tutorial21.c will show you how this is utilized.

```
#include<stdio.h>
#include<stdlib.h>


/*
Tutorial 21

This tutorial will show how to read input from the keyboard.
*/

int main(void)
{

int my_number;
double my_double;
char my_text[200];

// the "start_over" is used with the goto later in the function
start_over:


/* ask for an integer */

   fprintf(stdout,"Give me any integer (then hit return)\n");

/* read the integer from the keyboard */

   scanf("%d",&my_number);

/* print out the result */

   fprintf(stdout,"Good - The number you entered is %d\n",my_number);

/* lets make some spaces */

   fprintf(stdout,"\n\n\n");

/* ask for an double */

   fprintf(stdout,"Give me any real number (then hit return)\n");
```

```
/* read the integer from the keyboard */
/* make sure the format is "lf" */

   scanf("%lf",&my_double);

/* print out the result */

   fprintf(stdout,"Good - The number you entered is %f\n",my_double);

/* lets make some spaces */

   fprintf(stdout,"\n\n\n");

/* now ask for a string */

   fprintf(stdout,"Give me lip (I mean text), then hit return\n");

/* read in the string using scanf */

   scanf("%s",my_text);

/* print out the result */

   fprintf(stdout,"You wrote %s\n",my_text);

/* now, try this */

try_again:


   fprintf(stdout,"\n\n\n");
   fprintf(stdout,"Do you want to go again (y/n)?\n");

   scanf("%s",my_text);

   if (my_text[0]=='y')
   {
     goto start_over;
   }
   else if (my_text[0]=='n')
   {
     exit(10);
   }
   else
   {
     fprintf(stdout,"invalid input\n");
     goto try_again;
   }

return;
}
```

Switch Statements

Switch statements are useful for mainly two reasons. 1) They can replace if/then statements in a more concise way, and 2) they are used to provide a starting point to a sequence of commands. One thing to note, is that an argument in a switch statement should be an integer. Its best to learn by example: *Tutorial22.c*

```
#include<stdio.h>
#include<stdlib.h>


/*
Tutorial 22

This program helps explain the use of switch statements.
*/

int main(void)
{

int my_integer;

   fprintf(stdout,"Give me any number between 1 and 5\n");

   scanf("%d",&my_integer);

   fprintf(stdout,"The number you entered is %d\n",my_integer);

/*
In this first switch statement, we'll see how
"switch" can be used to tell where to start a
sequence of events.
Note the "default" expression.
Also note the "break command"
*/

   switch(my_integer)
   {
      case 1:
            fprintf(stdout,"Number1\n");
      case 2:
            fprintf(stdout,"Number2\n");
      case 3:
            fprintf(stdout,"Number3\n");
      case 4:
            fprintf(stdout,"Number4\n");
      case 5:
            fprintf(stdout,"Number5\n");
            break;
      default:
           fprintf(stdout,"Wrong number dum dum\n");

   }
```

```
/*
In this switch statement, we'll
see how switch can be used as a simple if/then.
Note, the "break" command is needed.
*/
   switch(my_integer)
   {
      case 1:
            fprintf(stdout,"Number 1 is a good number\n");
            break;
      case 2:
            fprintf(stdout,"Number 2 is my favorite\n");
            break;
      case 3:
            fprintf(stdout,"Number 3 is ok\n");
            break;
      default:
           fprintf(stdout,"I am bored already\n");
           break;
   }

return;
}
```

Function Pointers (ughh)

Now, we'll deal with pointers to a function. The main practical purpose of doing this is if you want to send a function to a function. Why would you ever want to do this? Well, imagine that you made a function called "trapezoid". Say your "trapezoid" function will return the value of the trapezoid integration. But, you may want to give several mathematical functions to "trapezoid". You can sent a pointer to a function to a function. Again, better seen in example:

Tutorial 23.c **[This tutorial will help you on a homework]**

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

/*
tutorial 23

This Tutorial helps to gain an understanding of pointers to functions

Note, this code contains the main function and a function named trap()
*/

int main(void)
{

/* DECLARATIONS */
double trap();
```

```
double result;

/* Look how a function pointer is declared! */
double (*my_pointer)();

/* Lets point the pointer to the cosine function and
   then run it through trap
*/

   my_pointer=cos;
   result=trap(M_PI/2.0, 0.0, 100, my_pointer);
   fprintf(stdout,"Result of cosine integration: %f\n",result);

/* Lets point the pointer to the exp function and
   then run it through trap
*/

   my_pointer=exp;
   result=trap(2.0, 0.0, 100, my_pointer);
   fprintf(stdout,"Result of exp integration: %f\n",result);

/* Lets point the pointer to the square root function and
   then run it through trap
*/

   my_pointer=sqrt;
   result=trap(3.0, 0.0, 100, my_pointer);
   fprintf(stdout,"Result of sqrt integration: %f\n",result);

return 0;
}

/* FUNCTION TRAP */
/* This function returns the trapezoid integration of the function
   pointed to by f_pointer.  The bounds are given as upper_limit and
   lower_limit.  The number of intervals that the integration is based on
   is given by n.
*/

double trap(upper_limit,lower_limit,n,f_pointer)
    double upper_limit;
    double lower_limit;
    int n;
    double (*f_pointer)();
{

double value;
double prefactor;
double h;
double x;
double increment;
int kk;

      increment=(upper_limit-lower_limit)/(1.0*n);
      h=fabs(increment);
```

```
      value=0.0;
      x=lower_limit;
      for (kk=0;kk<=n;kk++)
      {

/* find trapezoid rule prefactor */
         if ((kk==0)||(kk==n))
         {
            prefactor=1.0;
         }
         else prefactor=2.0;

         value=value+h/2.0*prefactor*(*f_pointer)(x);

         x=x+increment;
      }


return value;
}
```

Pointer to function arrays [This will help you on next homework]

We can also make arrays of pointers to functions.

Tutorial24.c

```
#include<stdio.h>
#include<stdlib.h>
/*
Tutorial 24 - arrays of pointers to functions
*/

int main(void)
{

double x;
double my_result;
double my_function1();
double my_function2();
double my_function3();
double my_function4();
int number_of_functions=4;
int kk;

/* here I define an array of pointers to functions */
double (*my_pointer[5])();

/* here, I define my pointer array to specific functions */

    my_pointer[1]=my_function1;
    my_pointer[2]=my_function2;
    my_pointer[3]=my_function3;
    my_pointer[4]=my_function4;
```

```
    x=5.0;

/* we set x=5.0, now, lets loop through all the functions,
   evaluating each one for x=5.0
*/

    for (kk=1;kk<=number_of_functions;kk++)
    {
       my_result=(*my_pointer[kk])(x);
       fprintf(stdout,"Result of x=%f for function %d is: %f\n",x,kk,my_result);

    }

return 0;
}

double my_function1(x)
   double x;
{
double value;

   value=x+1;

return value;
}

double my_function2(x)
   double x;
{
double value;

   value=2*x+2;

return value;
}

double my_function3(x)
   double x;
{
double value;

   value=3*x+3;

return value;
}

double my_function4(x)
   double x;
{
double value;

   value=4*x+4;

return value;
}
```