Computer Science 1510

Lecture 22

Lecture Outline

- Input / Output
- Selection
- Repetition

Basic Input and Output

- We have seen the `printf` function which allows the printing of formatted output to standard out (the screen).

- Now we will consider basic input, that is, accepting (or reading) data from the user.

- For this purpose we will use another function accessible via `stdio.h`, called `scanf`.

Basic Input in C: scanf

- `scanf` in C is similar to a `READ` statement in Fortran, in that it reads in data from the user into a variable, or variables.

- Like `printf`, `scanf` has the formatting built-in, and thus does not require the use of a separate format statement.

- Syntax:

  ```
  scanf("description",&variable-list);
  ```

  where `description` is a string describing the format of the input, and `variable-list` is a list of variables (each preceded by an ampersand &) where the input data should be stored.

- The `description` for `scanf` uses the same escape sequences and conversion codes as `printf`.

Basic Input in C: scanf

- As for `printf`, the number and type of variables in the `variable-list` must match the number and type of conversion codes in the `description`.

- The ampersand, &, is a new operator, which is prepended to a variable name to obtain the memory address of that variable.

- It is the address of a variable that must be passed to `scanf`.

- Thus, each variable in the `variable-list` must have an ampersand prepended to it, provided it is not an array. We will discuss array addresses later.

- Examples:

```
scanf("%d",&a);  /* Read in a single integer */
scanf("%c",&initial);  /* Read in a single character */
scanf("%f",&x);  /* Read in a single real value */
```

Example: scanf

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int age;
   float weight;
   char initial;

   printf("Enter your age, weight, and first initial\n");
   scanf("%d %f %c",&age,&weight,&initial);
   printf("Age: %d, Weight: %5.1f, Initial: %c\n",
           age,weight,initial);
   return 0;
}
```

A useful C compiler flag

- Compilers often have many flags which can be used to modify the compilation behaviour.

- We have seen some of these. For example, the `-o` flag allows one to specify the name of the executable.

- One very useful flag in gcc is `-W`, which is used to request the printing of warnings.

- Warnings are similar to error messages, but are not necessarily things that will cause undesired behaviour or results.

- The `-W` flag takes an extra argument to specify which warnings you would like to see.

- The most commonly used argument is `all` which says to display all warnings. So the compile command would be

```
gcc -Wall myprog.c
```

C: Selection

- In C, like in Fortran, there are two types of statements that can be used for selection.
- In C, these are the `if` statement, and the `switch` statement.
- The `switch` statement in C is similar to the `SELECT` statement in Fortran.
- We will consider each of these in more detail.

Selection: if statement

- Syntax:

  - Form 1:

```
if (logical_exp) statement;
```

  - Form 2:

```
if (logical_exp)
{
   statement-sequence;
}
```

  - Form 3:

```
if (logical_exp1){
   statement-sequence1;
}
else if (logical_exp2){
   statement-sequence2;
}
else{
   statement-sequence3;
}
```

Selection: if statement

- The `logical_exp` above is any valid C logical expression.

- Note that since C interprets integer values as true or false, any valid C arithmetic expression could also be used as the condition.

- The logical `?` operator can be used like an if statement. For example,

```
minimum = (a<b)? a : b
```

  is the same as writing,

```
if (a<b){
   minimum=a;
}
else {
   minimum=b;
}
```

Example 1: if and scanf

```
#include <stdio.h>
int main(int argc, char *argv[])
{
   short age,legal=19;
   printf("Please enter your age\n");
   scanf("%hd",&age);
   if (age>=legal) printf("You can enter\n");
   else printf("You are underage\n");
   return 0;
}
```

- Note that the conversion code for a `short` is `%hd`.
- Also, the conversion code for a `long` is `%ld`, and the conversion code for a `long double` is `%L`.

Example 2: if and scanf

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

int main(int argc, char *argv[]){
    float a, b, c;
    float dis, imag, real, x1, x2;

    printf("To calculate the roots of polynomial ax^2+bx+c,");
    printf("enter values for a, b, and c\n");
    scanf("%f %f %f",&a,&b,&c);

    dis = b*b - 4*a*c;

    if (dis > 0){  /* Equation has 2 distinct real roots */
        x1 = (-b+sqrt(dis))/(2*a);
        x2 = (-b-sqrt(dis))/(2*a);
        printf("The roots are %f and %f\n",x1,x2);
    }
    else if (dis < 0){  /* Equation has complex roots */
        real = -b/(2*a);
        imag = sqrt(fabs(dis))/(2*a);
        printf("The roots are %f+%fi and %f-%fi\n",real,imag,real,imag);
    }
    else if (dis==0){  /* Equation has a double root */
        x1 = -b/(2*a);
        printf("The equation has a double root at %f\n",x1);
    }

    return 0;
}
```

Selection: switch

- We have seen the use of `if` statements in C, we will now consider `switch` statements, which are analogous to SELECT statements in Fortran.

- A `switch` statement is used when one of several actions is to be taken depending on the value of a particular expression.

- Syntax:

```
switch (expression) {
    case value1:
       statement-sequence1;
       break;
    case value2:
       statement-sequence2;
       break;
    default:
       statement-sequenceD;
       break;
}
```

Selection: switch

- The result of the `expression` must be of type `char`, `short`, `int`, or `long`.
- This result is tested against `value1`, `value2`, etc., in turn, until a match is found. Execution continues with the statements following the matching case.
- Note that braces are not required for the `case` portions of the statement.
- The `break` statement is used to "break out of" the `switch` statement, such that execution continues from the statement following the `switch` block.
- If no match is found, execution will continue from the statement following the `default` clause.
- The `default` block is optional.

Selection: switch

- In other words, `case` statements indicate locations to begin execution within a `switch` block, while `break` statements indicate where to break out of the `switch` block.

- The `break` statement can be used within any C construct (such as an `if` statement, `switch` statement, or loop) to "break out of" the statement, and begin executing from the statement immediately after the construct.

Advantages and disadvantages of switch

- Advantages:
  - Can be easier to read than an `if` statement.
  - More efficient than an `if` statement (only requires a table look-up).

- Disadvantages:
  - Does not work with non-integer data types (such as `floats` or strings).
  - Does not work with ranges (unlike the Fortran `SELECT` statement).
  - Cannot use variables in the `case` values.

Example 1: switch

```
#include <stdio.h>
int main (int argc, char *argv[]){
   float k,m,result;
   char op;

   printf("Please enter expression to evaluate\n");
   scanf("%f %c %f",&k,&op,&m);

   switch (op) {
      case '+':
         result=k+m;
         break;
      case '-':
         result=k-m;
         break;
      case '*':
         result=k*m;
         break;
      case '/':
         result=k/m;
         break;
      default:
         printf("Invalid operator\n");
   }

   printf("%6.2f %c %6.2f = %8.2f\n",k,op,m,result);
   return 0;
}
```

Example 2: switch

```
#include <stdio.h>
int main(int argc,char *argv[]) {
   char c;
   printf("Enter a or b\n");
   scanf("%c",&c);
   switch (c) {
      case 'A':
      case 'a':
         printf("A ok!\n");
         break;
      case 'B':
      case 'b':
         printf("2B or !2B\n");
         break;
      default:
         printf("I said a or b, not %c!\n",c);
         break;
   }
   return 0;
}
```

Repetition

- There are three types of loops in C: a `while` loop, a `do while` loop, and a `for` loop.

- The `while` loop is analogous to the `DO WHILE` in Fortran, while the `for` loop is somewhat analogous to the counter controlled `DO` loop in Fortran.

- Unlike a `DO` loop in Fortran, which iterates for a fixed number of iterations, all loops in C will continue to iterate as long as their condition is true.

- We will look at each of these loops in detail.

while loop

- Syntax:

```
while (logical-expression){
    statement-block;
}
```

  where `logical-expression` is any valid C expression that evaluates to an integer that can be interpreted as true (non-zero) or false (zero).

- Example:

```
while (x<10 && y!=5){
    scanf("%d",&y);
    x++;
}
```

Example 1: while loop

Euclid's algorithm:

```
#include <stdio.h>
int main (int argc, char *argv[])
{
   int m,n,q,r;
   printf("Please enter two integers M and N, where M>N\n");
   scanf("%d %d",&m,&n);
   r=1;
   while (r!=0){
      q=m/n;
      r=m-q*n;
      printf("%d = %d * %d + %d\n",m,n,q,r);
      m=n;
      n=r;
   }
   printf("The greatest common divisor is %d\n",m);
   return 0;
}
```

for loop

- Syntax:

```
for (start-condition; end-condition; update){
   statement-block;
}
```

- Example:

```
for (i=0; i<10; i++){
   scanf("%d",&y);
   printf("%d",y);
}
```

- Note that the `for` loop begins counting at `i=0`. The default in C is to count from zero since array indices start from 0. (Fortran starts from 1).