**while Loop**

*Motivation*

There are some types of situations where we want repetition, but we don't know exactly how many times we want to repeat a set of steps. In these situations, a while loop is desireable. Consider the following game you probably played as a child: You guess a number from 1 to 100 that someone else is thinking of. Whenever you make a guess, they tell you higher or lower. You continue guessing until you get the number.

Depending on who is playing the game, it could run a different number of turns. In essence, what controls the end of the game isn't the number of turns, but whether or not you've guessed the correct number.

The while loop encapsulates this type of situation best. Here is the general construct of the while loop:

while (<boolean expression>) {

 stmt1;

 stmt2;

 ...

 stmtn;

}

Here is how the computer executes a while loop:

1) The Boolean expression is evaluated.

2) If it’s true, we execute stmt1 though stmtn. Then we proceed back to 1.

3) If the Boolean expression is false, we are done with the statement and continue execution after the while loop.

For our situation of the guessing game, we want the game to continue so long as the user's guess is NOT equal to the secretly chosen number. The code for the number game is included on the following page:

#include <stdio.h>

#include <time.h>

int main(void) {

 int secret, guess = -1;

 int numguesses = 0;

 srand(time(0));

 secret = 1 + rand()%100;

 while (guess != secret) {

 printf("Enter your next guess.(1 to 100)\n");

 scanf("%d", &guess);

 numguesses++;

 if (guess < secret) {

 printf("You need to guess higher. Try again!\n");

 }

 else if (guess > secret) {

 printf("You need to guess lower. Try again!\n");

 }

 }

 printf("You got it in %d guesses.\n", numguesses);

 return 0;

}

In fact, any program written with a for loop can also be written with a while loop. Let's take a look at the following code segment from the for loop notes:

 int sum = 0, count = 1;

 for (count=1; count <= MAX\_TERM; count=count+1) {

 sum = sum + count;

 }

We can rewrite as follows:

 int sum = 0, count = 1;

 while (count <= MAX\_TERM) {

 sum = sum + count;

 count=count+1;

 }

Another standard program archetype that is solved with the for loop is a menu driven program. Consider a situation where we want to give the user a few choices and they can continue choosing what they want to do until they quit. A simple example of such a program is a bank program, where we allow the user to deposit and withdraw money from an account. In this program we will give the user 4 options:

1) Deposit money

2) Withdraw money

3) Print their balance

4) Quit

We will use one variable to store the user's balance and when necessary we'll use a second variable to read in a user's deposit or withdrawal. Let's look at the general construct:

// Print menu

scanf("%d", &choice);

while (choice != QUIT) {

 if (choice == 1) {

 // Execute choice 1

 }

 else if (choice == 2) {

 // Execute choice 2

 }

 ...

 else if (choice != QUIT) {

 // Error message

 }

 // Print menu

 scanf("%d", &choice);

}