Flow Control

Comments

- Comments: /* This is a comment   */
  - Use them!
  - Comments should explain:
    - special cases
    - the use of functions (parameters, return values, purpose)
    - special tricks or things that are not obvious
  - explain *WHY* your code does things the what it does.

More on Comments

- A bad comment:

```
…
i = i + 1;                    /* assign i+1 to the variable i */
…
```

- A better comment:

```
…
i = i + 1;                    /* increment the loop counter */
…
```

C++ Comments

- A bad comment:

```
…
i = i + 1;                    // assign i+1 to the variable i
…
```

- A better comment:

```
…
i = i + 1;                    // increment the loop counter
…
```

C Statements

- In the most general sense, a statement is a part of your program that can be executed.
- An expression is a statement.

```
a=a+1;
a--;
```

- A function call is also a statement.

```
printf("%d",a);
```

- Other statements ……
- C is a free form language, so you may type the statements in any style you feel comfortable:

```
a=
a+
1;a--;
```

line breaks can be anywhere

Compound Statements

- Sequences of statements can be combined into one with {...}
- Much like Java:

```
{
    printf ("Hello, ");
    printf ("world! \n");
}
```

- The C compiler treats the collection of these statements like they are a single statement.

C Statements

Some Suggestions

- DO: stay consistent with how you use whitespace

- DO: put block braces on their own line.
  - This makes the code easier to read.

- DO: line up block braces so that it is easy to find the beginning and end of a block.

- AVOID: spreading a single statement across multiple lines if there is no need.
  - Try to keep it on one line.

The if Statement (1)

- Form 1:

```
if (expression)
        statement1;
next statement;
```

Execute statement1
if expression is non-zero
(i.e., it does not have to be exactly 1)

- Form 2:

```
if (expression)
        statement1;
else
        statement2;
next statement;
```

- Form 3:

```
if (expression)
        statement1;
else if (expression)
            statement2;
else
            statement3;
next statement;
```

The if Statement (2)

- For Example:

```
#include <stdio.h>
int x,y;
int main ()
{
        printf ("\nInput an integer value for x: ");
        scanf ("%d", &x);
        printf ("\nInput an integer value for y: ");
        scanf ("%d",&y);
        if (x==y)
                printf ("x is equal to y\n");
        else if (x > y)
                printf ("x is greater than y\n");
        else
                printf ("x is smaller than y\n");
        return 0;
}
```

The for Statement (1)

- The most important looping structure in C.
- Generic Form:

  for (initial ; condition ; increment)

  statement

- *initial*, *condition*, and *increment* are C expressions.
- For loops are executed as follows:

  1. *initial* is evaluated. Usually an assignment statement.
  2. *condition* is evaluated. Usually a relational expression.
  3. If *condition* is false (i.e. 0), fall out of the loop (go to step 6.)
  4. If *condition* is true (i.e. nonzero), execute *statement*
  5. Execute *increment* and go back to step 2.
  6. Next statement

The for Statement (2)

For statement examples

```
#include <stdio.h>
int main () {
   int count,x,y;
   int ctd;

   /* 1. simple counted for loop */
   for (count =1; count <=20; count++)
       printf ("%d\n", count);

   /* 2. for loop counting backwards */
   for (count = 100; count >0; count--) {
      x*=count;
      printf("count=%d x=%d\n", count,x);
   }

   /* 3. for loop counting by 5's */
   for (count=0; count<1000; count += 5)
         y=y+count;

   /* 4. initialization outside of loop */
   count = 1;
   for ( ; count < 1000; count++)
      printf("%d ", count);

   /* 5. very little need be in the for */
   count=1; ctd=1;
   for ( ; ctd; ) {
       printf("%d ", count);
       count++; ctd=count<1000;
   }

   /* 6. compound statements for
     initialization and increment */
   for (x=0, y=100; x<y; x++, y--) {
      printf("%d %d\n", x,y);
   }
   return 0;
}
```

The for Statement (3)

- Nesting for Statements
  - for statements (and any other C statement) can go inside the loop of a for statement.
  - For example:

```
#include <stdio.h>
int main( ) {
   int rows=10, columns=20;
   int r, c;
   for ( r=rows ; r>0 ; r--)
   {
       for (c = columns; c>0; c--)
          printf ("X");
       printf ("\n");
    }
}
```

The while Statement

- Generic Form

```
while (condition)
    statement
```

- Executes as expected:
  1. condition is evaluated
  2. If condition is false (i.e. 0), loop is exited (go to step 5)
  3. If condition is true (i.e. nonzero), statement is executed
  4. Go to step 1
  5. Next statement
- Note:
  - `for ( ; condition ; )` / `stmt;` is equivalent to `while (condition)` / `stmt;`

```
for ( ; condition ; )          while (condition)
    stmt;                          stmt;
```

  - `for (exp1; exp2; exp3) stmt;`

    is equivalent to

```
exp1;
while(exp2) { stmt; exp3;  }
```

The do ... while Loop (1)

- Generic Form:

```
do
    statement
while (condition);
```

- Standard repeat until loop
  - Like a while loop, but with condition test at bottom.
  - Always executes at least once.
- The semantics of do...while:

1. Execute statement
2. Evaluate condition
3. If condition is true go to step 1
4. Next statement

The do ... while Loop (2)

```
#include <stdio.h>
int get_menu_choice (void);
main()
{
  int choice;
  do
  {
    choice = get_menu_choice ();
    printf ("You chose %d\n",choice);
  } while(choice!=4);
  return 0;
}
```

```
/* simple function get_menu_choice */

int get_menu_choice (void)
{
  int selection = 0;
  do {
    printf ("\n");
    printf ("\n1 - Add a Record ");
    printf ("\n2 - Change a Record ");
    printf ("\n3 - Delete a Record ");
    printf ("\n4 - Quit ");
    printf ("\n\nEnter a selection: ");
    scanf  ("%d", &selection);
  } while ( selection<1 || selection>4);
  return selection;
}
```

break and continue

- The flow of control in any loop can be changed through the use of the break and continue commands.
- The break command exits the loop immediately.
  - Useful for stopping on conditions not controlled in the loop condition.
  - For example:

```
for (x=0; x<10000; x++) {
   if ( x*x % 5==1) break;
   ... do some more work ...
}
```

  - Loop terminates if x*x % 5 == 1
- The continue command causes the next iteration of the loop to be started immediately.
  - For example:

```
for (x=0; x<10000; x++) {
   if (x*x % 5 == 1) continue;
   printf( "%d ", 1/ (x*x % 5 – 1) );
}
```

  - Don't execute loop when x*x % 5 == 1 (and avoid division by 0)

Example: for and break Together

```
const int mycard=3;
int guess;
for(;;)
{
    printf("Guess my card:");
    scanf("%d",&guess);
    if(guess==mycard)
    {
        printf("Good guess!\n");
        break;
    }
    else
        printf("Try again.\n");
}
```

The notation for(;;) is used to create an infinite for loop. while(1) creates an infinite while loop instead.

To get out of an infinite loop like this one, we have to use the break statement.

switch Statement

- Switch statement is used to do “multiple choices”.
- Generic form:

```
switch(expression)
{
    case constant_expr1 : statements
    case constant_expr2 : statements
    …
    case constant_exprk : statements
    default : statements
}
```

1. expression is evaluated.
2. The program jumps to the corresponding constant_expr.
3. All statements after the constant_expr are executed until a break (or goto, return) statement is encountered.

Example: switch Statement

```
int a;
printf("1. Open file..\n");
printf("2. Save file.\n");
printf("3. Save as..\n");
printf("4. Quit.\n");
printf("Your choice:");
scanf("%d", &a);
if(a==1)
    open_file();
else if(a==2)
    save_file();
else if(a==3)
    save_as();
else if(a==4)   return 0;
else  return 1;
```

```
int a;
printf("1. Open file..\n");
printf("2. Save file.\n");
printf("3. Save as..\n");
printf("4. Quit.\n");
printf("Your choice:");
scanf("%d", &a);
switch(a)
{
    case 1: open_file();break;
    case 2: save_file();break;
    case 3: save_as();break;
    case 4: return 0;
    default: return 1;
}
```

Jumping Out of Nested Loops -- goto

- The goto statement will jump to any point of your program
- Use only if it is absolutely necessary (never in this course)

```
for(;;)
{
    ……
    while(…)
    {
       switch(…)
        {
           ……
           case … : goto finished;    /* finished is a label */
       }
    }
}
finished:    /* Jumped out from the nested loops */
```

Never jump into a loop!
Never jump backward!