San Lorenzo High School C Programming Class

Lesson of the Day

Consider the following code:

```
#include <stdio.h>

char *Message[] =     { "All Problems are solvable",
                        "There are no bugs in TASM",
                        "If you need something done do it yourself"};


 void main(void)
 {
    int line;

    for (line = 0; line<3; line++)
     printf("LINE:[%d] - %s\n", line, Message[line]);

 }
```

Please try to describe what the code will do before you type it in and run it.

Class Assignment

Create a program that displays both the number of command-line arguments and each command line argument that is passed to **main**().

You main function should look like the following:

```
void main(int argc, char *argv[])
{
      // Add code in here to print the total command-line
      // arguments and each argument.

}
```

Advanced Class Assignment

Use the functions defined/prototyped in **CONIO.H** to create a program that accepts three command-line arguments. The first two command-line arguments will specify a location on the screen and the last or third command-line argument will specify a string to print at the given location. Consider looking in the Run-Time Library Reference Manual at the function *cprintf()*, *gotoxy()*, and *atoi()*. These functions will aid in the creation of the program.

Program #1:

Write a program that accepts single characters at a time using getch(). Using a switch statement you should print out the following strings depending on what keys are pressed:

<UP>	"Up"
<DOWN>	"Down"
<LEFT>	"Left"
<RIGHT>	"Right"
<ENTER>	"Enter"

If you press <Esc> then the program should end.

The list of keys is :

<UP>	0x48
<DOWN>	0x50
<LEFT>	0x4B
<RIGHT>	0x40
<Escape>	27

Example to get you started:

```
#include <conio.h>

int main (void)
{
  int c = 0;
  while (1)
  {
    c = getch ();
    if (c == 0)
      c = getch ();
    switch (c)
    {
      case 27:
        exit (0);
        break;
      case 0x48:
        // print the string "Up" here
        break;
    }
  }
}
```

Program #2:

Add to program #1 so that you print out the current values of an 'x' and a 'y' coordinate instead of the "Up", "Down"... message. The x and y values should be initialized to (1,1). Every time you press the <UP> arrow, you should subtract 1 from the y value. Every time you press the <RIGHT> arrow, you should add 1 to the x value and so on. This should allow you to move the coordinate values from (1,1) to (80,25). Make sure that the values don't go over the maximum limit or below the minimum limit.

Program #3:

Add to program #2 so that instead of printing the current locations of the 'x' and 'y' coordinates, you move to that location on the screen using gotoxy() and then put a star "*" on that spot on the screen using cprintf. You should also clear the screen of the old star using clrscr() before printing the new one. At this point you should be able to move the star all over the screen using the arrow keys.

Program #4:

Add this to program #3. Write a new function called MyGetch that has the following prototype:

```
int MyGetch (void);
```

This function should look like this:

```
int MyGetch (void)
{
  if (!kbhit())
  {
    UpdateBackground ();
  }
  return getch ();
}
```

You should also prototype a function called UpdateBackground that looks like this:

```
void  UpdateBackground (void);
```

For now, this function should do nothing.

Program #5:

Add this to program #4. Implement the function UpdateBackground. This function should keep a unique set of x and y coordinates that will display a dollar sign "$" where the current location is. Every time the location function is called, you should update the coordinates by adding a vector to them. Normally this vector will start at (+1,+1) so every time the UpdateBackground functions is called a 1 will be added to the x value and a 1 to the y value. If either of the coordinates hit the boundaries (ie the maximum or minimum values for that axis: 1-25 for y and 1-80 for x) then multiply that vector's axis by -1. So every time the "$ " hits a wall it bounces off. This functionality should remain independent from the cursor movement that you have implemented in the previous programs.

Program #6:

Add this to program #5. Every time the "$' hits a wall and changes direction, you should randomly change that axis value in the vector to a number from 1-4 using the same sign ie: if the "$" just hit a wall and the vector is currently (-1,1) and you are going to change the vector for the x axis, pick a random number, say 3, and now assign the sign of the current vector to it (-3). The vector is now (-3,1) and the "$" moves much faster in the x direction. Make sure that the vectors never exceed (+-4,+-4) and make sure that they are never 0.

Program #7:

Add this to program #6. Now make the enter key in the main part of the loop leave a screen dropping in the location where it was pressed. The character that it leaves should be the "@" sign. Now instead of clearing the screen each iteration through the loop (with clrscr()) you need to print a space at the old location before writing the "*" in the new location. Same with the "$" part in the UpdateBackground function. Now the only things that should stay on the screen are the one current "$", the one current "*" and however many "@" there are. Now change the UpdateBackground function to not only reverse the vector's directions upon impact with a wall, but also if it hits a "@".