BIT FIELDS:

A more compact method of manipulating bits is by using a bit field. A bit field is a number of bits that are all located within an unsigned int type. You must set up the bit field with a structure definition that labels each field and also specifies the bit width for each field. The sum of the widths for all of the labels must not exceed the bit width for the unsigned int type. You can also pad a field structure with unnamed holes by using unnamed field widths.

```
For Example,
#include <stdio.h>

#define SYSTEM_DEFAULT        0x7
#define POWER_ON_MASK         0x1
#define HEALTH_OFF_MASK       0x0
#define HEALTH_YELLOW_MASK    0x1
#define HEALTH_RED_MASK       0x2
#define HEALTH_GREEN_MASK     0x3
#define BUSY_MASK             0x1

struct PanelLeds
{
      /* 0=off, 1=green; This is also the LSB */
      unsigned power : 1;

      /* 00=off, 01=yellow, 10=red, 11=green  */
      unsigned systemHealth : 2;

      /* Pad bits to fill lower byte*/
      unsigned : 5;

      /* 0=off, 1=red */
      unsigned busy : 1;
};

union Leds
{
      struct PanelLeds  sPanelLeds;
      unsigned          uPanelLeds;
} Leds;
```

```
int main(void)
{
      /* Set up the initial conditions for the system */
      Leds.sPanelLeds.power |= POWER_ON_MASK;
      Leds.sPanelLeds.systemHealth |= HEALTH_GREEN_MASK;
      Leds.sPanelLeds.busy &= ~BUSY_MASK;

      /* Print out the initial conditions */
      printf("\nINITIAL CONDITIONS\n");
      printf("\tPower = %#x\tHealth = %#x\tBusy = %#x\n",
                                    Leds.sPanelLeds.power,
                                    Leds.sPanelLeds.systemHealth,
                                    Leds.sPanelLeds.busy);
      printf("\tFull unsigned = %#x\n", Leds.uPanelLeds);

      /* Set health LED to yellow and busy to red */
      Leds.sPanelLeds.systemHealth  &= HEALTH_OFF_MASK;
      Leds.sPanelLeds.systemHealth  |= HEALTH_YELLOW_MASK;
      Leds.sPanelLeds.busy          |= BUSY_MASK;

      /* Print out the new system conditions */
      printf("\nNEWLY CHANGED CONDITIONS\n");
      printf("\tPower = %#x\tHealth = %#x\tBusy = %#x\n",
                                    Leds.sPanelLeds.power,
                                    Leds.sPanelLeds.systemHealth,
                                    Leds.sPanelLeds.busy);
      printf("\tFull unsigned = %#x\n", Leds.uPanelLeds);

      /*
       * Reset all bits to the default setting using the
       * uPanelLeds variable .
       */
      Leds.uPanelLeds &= 0x0;
      Leds.uPanelLeds |= SYSTEM_DEFAULT;

      /* Print out the new system conditions */
      printf("\nSYSTEM SET TO DEFAULT\n");
      printf("\tPower = %#x\tHealth = %#x\tBusy = %#x\n",
                                    Leds.sPanelLeds.power,
                                    Leds.sPanelLeds.systemHealth,
                                    Leds.sPanelLeds.busy);
      printf("\tFull unsigned = %#x\n", Leds.uPanelLeds);

      /*
       * Set the busy bit using uPanelLeds
       */
      Leds.uPanelLeds |= 0x100; /* busy bit is b8 */
```

```
        /* Print out the new system conditions */
        printf("\nSYSTEM BUSY CONDITION\n");
        printf("\tPower = %#x\tHealth = %#x\tBusy = %#x\n",
                                        Leds.sPanelLeds.power,
                                        Leds.sPanelLeds.systemHealth,
                                        Leds.sPanelLeds.busy);
        printf("\tFull unsigned = %#x\n", Leds.uPanelLeds);

        return 0;
}
```

Sample Run,

```
Y:\apc-cc-conduit\Images>A

INITIAL CONDITIONS
        Power = 0x1     Health = 0x3    Busy = 0
        Full unsigned = 0x7

NEWLY CHANGED CONDITIONS
        Power = 0x1     Health = 0x1    Busy = 0x1
        Full unsigned = 0x103

SYSTEM SET TO DEFAULT
        Power = 0x1     Health = 0x3    Busy = 0
        Full unsigned = 0x7

SYSTEM BUSY CONDITION
        Power = 0x1     Health = 0x3    Busy = 0x1
        Full unsigned = 0x107

Y:\apc-cc-conduit\Images>
```

POINTERS

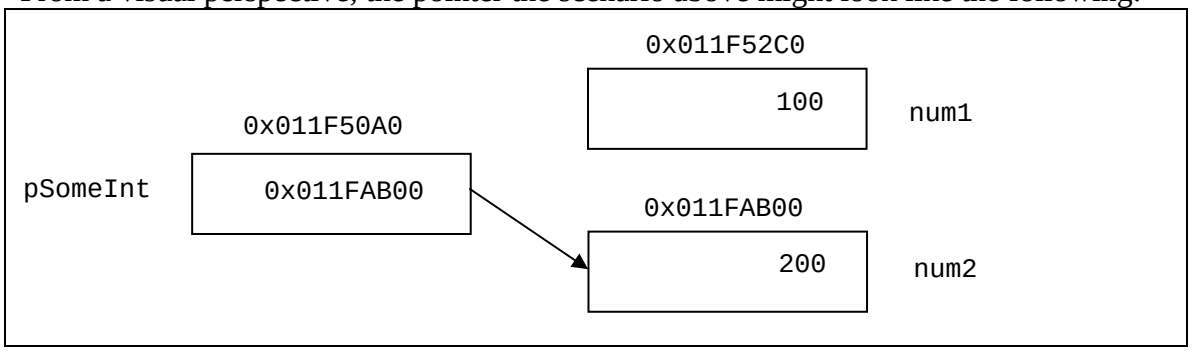
A pointer is a special variable whose purpose is to hold a memory address. A pointer variable is declared by specifying the pointer type, the indirection operator (*), the pointer name, and an optional pointer initialization.

Once created, the pointer can be used to hold a memory address of another variable of the same type. When a pointer holds the memory address of a variable of the same type, we say that the pointer *points* to the variable. In fact, a pointer can be re-assigned point to any variable of the same type. However, it can only point to one variable at a time.

For Example,

```
int num1, num2;
int *pSomeInt = NULL;    /* Declare an integer pointer */

pSomeInt = &num1; /* Pointer points to num1 */
pSomeInt = &num2; /* Pointer points to num2 */

OR

int num1, num2;
int *pSomeInt = &num1; /* Declare pointer that points to num1 */

pSomeInt = &num2;      /* Pointer points to num2 */

 OR

int num1 = 100, num2 = 200;
int *pSomeInt = &num1; /* Declare pointer that points to num1 */

pSomeInt = &num2;      /* Pointer points to num2 */
```

From a visual perspective, the pointer the scenario above might look like the following:



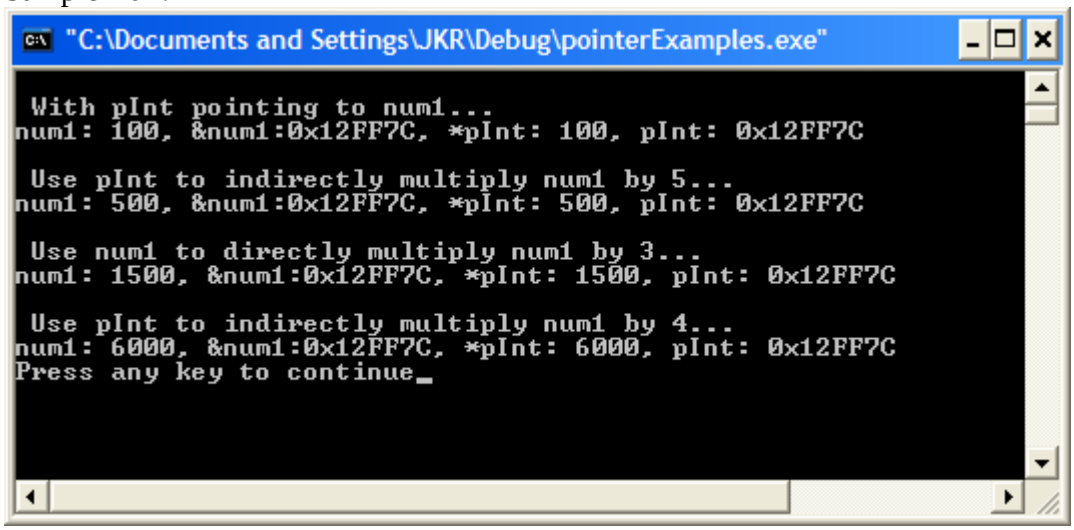
We say that `pSomeInt` points to `num2` in the figure above. Notice that the value of `pSomeInt` is the memory address of the `num2` variable. To make `pSomeInt` point to `num1`, we simply change the value of `pSomeInt` to the address of the `num1` variable. Some additional pointer examples are given below that show how to declare various types of pointers as well as how to access the data to which the pointers point.

It is important to remember that a pointer is a special variable that holds a memory address. The variable existing at that memory address is the variable to which the pointer points. Thus, in each case below, you should remember that the pointer is only pointing to one variable at a time.

Some Simple Examples:

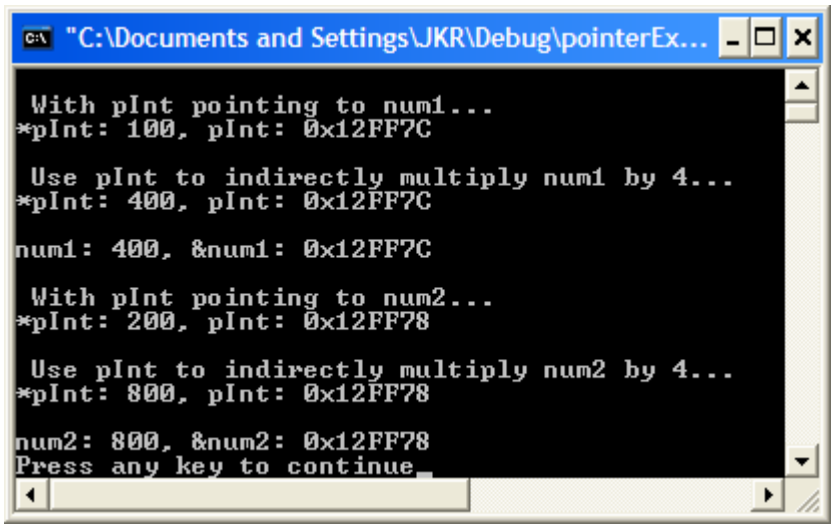
```
#include <stdio.h>

int main(void)
{
      int num1 = 100;
      int *pInt = NULL;


      pInt = &num1;
      printf("\n With pInt pointing to num1...");
      printf("\nnum1: %d, &num1:0x%X, *pInt: %d, pInt: 0x%X\n",
            num1, &num1, *pInt, pInt);

      printf("\n Use pInt to indirectly multiply num1 by 5...");
      *pInt = *pInt * 5;
      printf("\nnum1: %d, &num1:0x%X, *pInt: %d, pInt: 0x%X\n",
            num1, &num1, *pInt, pInt);

      printf("\n Use num1 to directly multiply num1 by 3...");
      num1 *= 3;
      printf("\nnum1: %d, &num1:0x%X, *pInt: %d, pInt: 0x%X\n",
            num1, &num1, *pInt, pInt);

      printf("\n Use pInt to indirectly multiply num1 by 4...");
      *pInt *= 4;
      printf("\nnum1: %d, &num1:0x%X, *pInt: %d, pInt: 0x%X\n",
            num1, &num1, *pInt, pInt);


      return 0;
}
```

Sample Run:



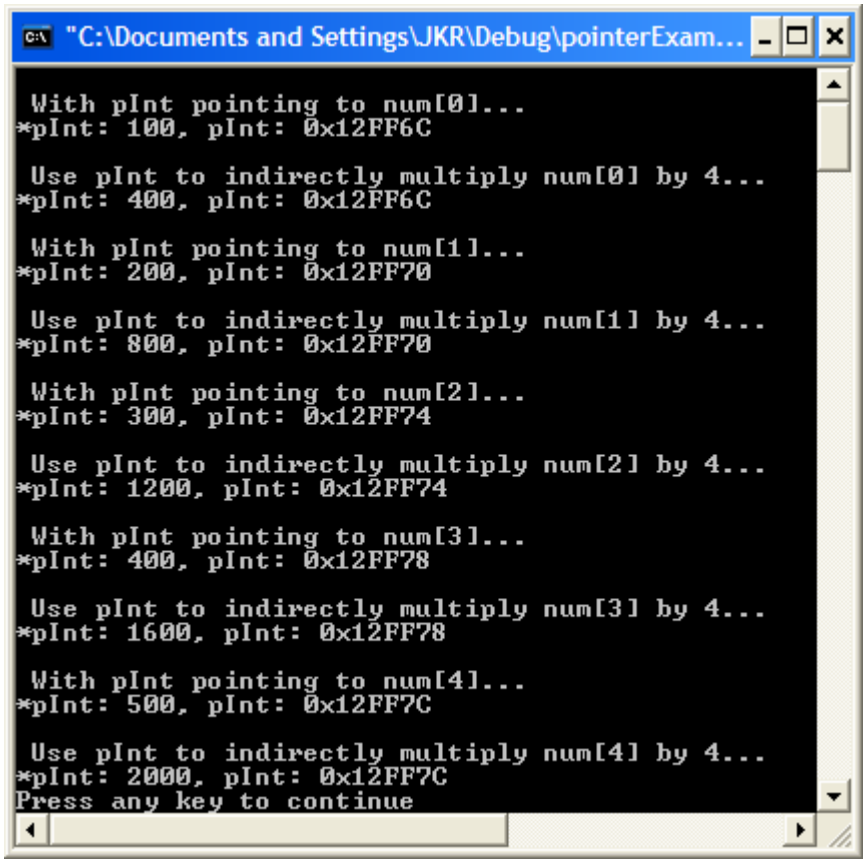
```
#include <stdio.h>

int main(void)
{
	int num1 = 100;
	int num2 = 200;
	int *pInt = NULL;
	int counter;

	for(counter = 0; counter < 2; counter++)
	{
		if(counter == 0)
			pInt = &num1;
		else
			pInt = &num2;

		printf("\n With pInt pointing to num%d...", counter == 0 ? 1 : 2);
		printf("\n*pInt: %d, pInt: 0x%X\n", *pInt, pInt);

		printf("\n Use pInt to indirectly multiply num%d by 4...",
			counter == 0 ? 1 : 2);

		*pInt *= 4;
		printf("\n*pInt: %d, pInt: 0x%X\n", *pInt, pInt);

		if(counter == 0)
			printf("\nnum1: %d, &num1: 0x%X\n", num1, &num1);
		else
			printf("\nnum2: %d, &num2: 0x%X\n", num2, &num2);
	}

	return 0;
}
```

OUTPUT



```
#include <stdio.h>

int main(void)
{
	int num[] = { 100, 200, 300, 400, 500 };
	int *pInt = NULL;
	int counter;

	for(counter = 0; counter < 5; counter++)
	{
		pInt = &num[counter];

		printf("\n With pInt pointing to num[%d]...", counter);
		printf("\n*pInt: %d, pInt: 0x%X\n", *pInt, pInt);

		printf("\n Use pInt to indirectly multiply num[%d] by 4...", counter);

		*pInt *= 4;
		printf("\n*pInt: %d, pInt: 0x%X\n", *pInt, pInt);
	}

	return 0;
}
```

OUTPUT



```
#include <stdio.h>

int main(void)
{
        int num[] = { 100, 200, 300, 400, 500 };
        int *pInt = NULL;
        int counter;

        for(pInt = num, counter = 0; counter < 5; counter++, pInt++)
        {
                printf("\n With pInt pointing to num[%d]...", counter);
                printf("\n*pInt: %d, pInt: 0x%X\n", *pInt, pInt);

                printf("\n Use pInt to indirectly multiply num[%d] by 4...", counter);

                *pInt *= 4;
                printf("\n*pInt: %d, pInt: 0x%X\n", *pInt, pInt);
        }

        return 0;
}
```

OUTPUT