**Bitsel İşlemler(Bitwise Operations)**

C programlama dilinde şimdiye kadar pek çok operatör öğrendik ancak henüz bitsel operatörleri hiç öğrenmedik. Bitsel operatörler gömülü yazılım başta olmak üzere kullanım alanı oldukça geniş olan operatörlerdir. C’de ve pek çok programlama dilinde altı adet bitsel operatör bulunmaktadır(bitsel işlemli atama operatörleri). Bunlar:

1-) Bitsel değil operatörü (Bitwise not operator(~)) : Operatör öncelik öncelik tablosunun ikinci öncelik seviyesinde bulunan bu operatör söz konusu değişkenin binary(ikilik) sistemdeki tüm bitlerini değiştiren(toggle eden) bir operatördür. Yan etkisi yoktur.

2-) Bitsel sağa kaydırma operatörü (Bitwise right shift operator(>>)) : Operatör öncelik tablosunun beşinci öncelik seviyesinde bulunan bu operatör sol operandı olan sayının binary sistemdeki halini sağ operand kadar sağa kaydıran bir operatördür. Yan etkisi yoktur.

3-) Bitsel sola kaydırma operatörü (Bitwise left shift operator(<<)) : Operatör öncelik tablosunun beşinci öncelik seviyesinde bulunan bu operatör sol operandı olan sayının binary sistemdeki halini sağ operand kadar sola kaydıran bir operatördür. Yan etkisi yoktur

4-) Bitsel ve operatörü (Bitwise and operator(&)) : Operatör öncelik tablosunun sekizinci öncelik seviyesinde bulunan bu operatör sağ ve sol operandında bulunan operandların bitleri birer birer ve işlemine sokulur. Yan etkisi yoktur.

5-) Bitsel veya operatörü (Bitwise or operator( | )) : Operatör öncelik tablosunun onuncu öncelik seviyesinde bulunan bu operatör sağ ve sol operandında bulunan operandların bitleri birer birer veya işlemine sokulur. Yan etkisi yoktur.

6-) Bitsel özel veya operatörü (bitwise exor(^)) : Operatör öncelik tablosunun dokuzuncu öncelik seviyesinde bulunan bu operatör iki sayıyı binary sistemde ele alarak aynı bitlerle farklı ise 1 eğer aynı ise 0 üreten bir operatördür. Yan etkisi

7-) Bitsel işlemli atama operatörleri(Bitwise operative assignment operatör (<<= >>= &= ^= |=)) : Operatör öncelik tablosunun on dördüncü öncelik seviyesinde bulunan bu operatörler bitsel operatörlerle yapılan işleri okunabilirlik ve algılanabilirlik açısından kodu daha iyi hale getirirler. Yan etkisi vardır.

Aşağıdaki test kodu kullanılarak bir değişken binary sistemde yazdırılabilir. Örneğin:

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val)

{

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main()

{

int val=10;

bitprint(val);

printf("\n");

bitprint(~val);

}

Bu örnekte bitsel değil operatörün de nasıl çalıştığı gösterilmiştir. Daha önce dediğimiz gibi bu operatör tüm bitleri değiştirmeye yarar.

Bitsel kaydırma operatörleri

1-) Bitsel sola kaydırma(<<)

2-) Bitsel sağa kaydırma(>>)

x << y => Bu ifade x sayısının bitlerini y kadar dola kaydır demektir.

**Anahtar Notlar :** Bitsel sola kaydırma işleminde sağ operandın negatif değer olması tanımsız davranışa neden olur. Örneğin :

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val)

{

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main()

{

int val=10;

bitprint(val);//0000 0000 0000 0000 0000 0000 0000 1010

printf("\n");

bitprint(val<<-1);//Undefined Behaviour

}

**Anahtar Notlar :** Bitsel sola kaydırma işleminde sağ operandın tamsayı türünün bit sayısından büyük olması tanımsız davranışa neden olur. Örneğin :

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val)

{

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main()

{

int val = 10;

bitprint(val);//0000 0000 0000 0000 0000 0000 0000 1010

printf("\n");

bitprint(val << 35);//Undefined Behaviour

}

**Anahtar Notlar :** Sağdan yapılacak besleme sol operandın türünden bağımsız olarak sıfırdır. Örneğin :

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val)

{

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main()

{

int val=10;

bitprint(val);//0000 0000 0000 0000 0000 0000 0000 1010

printf("\n");

bitprint(val<<2);//0000 0000 0000 0000 0000 0000 0010 1000

printf("\n");

bitprint(val);//0000 0000 0000 0000 0000 0000 0000 1010

}

Eğer bitsel operatörlerde bir side effect oluşması istenirse bu durumda işlemli atama operatörleri kullanılabilir. Örneğin :

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val)

{

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main()

{

int val=10;

bitprint(val);//0000 0000 0000 0000 0000 0000 0000 1010

printf("\n");

val<<=2;

bitprint(val);//0000 0000 0000 0000 0000 0000 0010 1000

}

**Anahtar Notlar :** İşaretsiz(unsigned) tamsayı türleri için her zaman besleme(feeding) sıfır ile yapılır. Örneğin :

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val)

{

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main()

{

unsigned uVal = 10; //0000 0000 0000 0000 0000 0000 0000 1010

bitprint(uVal << 2);// 0000 0000 0000 0000 0000 0000 0010 1000

}

**Anahtar Notlar :** İşaretli tamsayı türleri için besleme sol operandın işaretine göre değişir. Örneğin sol operand pozitif değerde ise her zaman besleme(feeding) sıfır(0) ile yapılır. Sol operand negatif ise derleyiciye bağlı olarak besleme sıfır(0) ya da bir(1) biti ile yapılır. Bu duruma sign extension adı verilir. Bu açıklamaya göre aşağıdaki örneği inceleyiniz.

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val)

{

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main()

{

int val=-1;

bitprint(val);

printf("\n");

val<<=-2;

bitprint(val);

}

Sağa kaydırma operatörünün özellikleri de sola kaydırma operatörüyle genel olarak aynıdır.

tek fark adından da anlaşılacağı gibi sayının bitlerini sola değil sağa

kaydırır. beslemeler sağ operandın pozitif olması durumunda daima 0 ile

yapılır.

**Anahtar Notlar :** Bitsel sağa kaydırma operatörüyle bir sayıyı iki defa sağa kaydırmak o sayıyı dörde bölmek arasında bir fark yoktur. Kısacası bölme işleminin bölen kısmı için iki üzeri kaydırma sayısı gibi bir genel formül söz konusu olabilir. Aynı durumun tersi sola kaydırma için de geçerlidir yani bir sayıyı iki defa sola kaydırma dört ile çarpmak anlamına gelmektedir. Bu durum aşağıdaki örnekte açıklanmıştır :

#include <stdio.h>

int main()

{

int val = 10;

printf("%d\n", val / 4 == val >> 2);

printf("%d\n", val \* 4 == val << 2);

}

**Bitsel ve operatörü(Bitwise and operator)**

Bu operatör adress of(ampersand) operatörü ile karıştırılmamalıdır. Ayırt etmenin en basit yolu daha önce operatörler konusunda da bahsettiğimiz gibi kullanım yerine bağlı olarak aldığı operand sayısına bağlıdır. Örneğin iki tane operand alıyorsa bu durumda binary kullanıma sahiptir. Binary kullanımda ise şüphesiz bu bitsel ve operatörüdür. Ancak unary kullanıma sahip ise bu durumda ampersand veya address of operatörüdür. Kullanımı lojik ve operatörüyle benzerlik gösterir. Aradaki en önemli fark bitlerin tek tek ele alınmasıdır. Bitsel ve operatörünün genel davranışı şu şekildedir:

En az bir bit 0 ise sonuç olarak 0, 2 bit de 1 ise 1 biti üreten bir operatördür. Kısaca lojik olanının tüm bitlere yayılmış halidir diyebiliriz. Bunlardan yola çıkarak aşağıdaki örneği inceleyiniz:

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val)

{

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main(){

int val1=20,val2=10;

bitprint(val1);//0000 0000 0000 0000 0000 0000 0001 0100

printf("\n");

bitprint(val2);//0000 0000 0000 0000 0000 0000 0000 1010

printf("\n");

if (val1 && val2)

printf("True for logical and\n");

if (val1 & val2)

printf("True for bitwise and");

else

printf("False for bitwise and");

}

Bu örnekte 10 ile 20 sayılarının farklı bitleri 1 olduğundan dolayı(hiç aynı bit 1 olmadığından dolayı) tüm bitler sıfırlanmış ve sonuç 0 olarak elde edilmiştir. Bu örnekte aynı zamanda bitsel ve ile lojik ve de birlikte kullanılmıştır.

**Bitsel Veya Operatörü(Bitwise or operator)**

Bu operatör de tahmin edilebileceği gibi lojik veya operatörüne benzemektedir. Aradaki fark bitsle ve ile lojik ve arasında fark ile bire bir aynıdır. Bitler birer birer ele alınırken lojik veyada olan o önemli kural devreye girer. “En az biri doğruyla doğru, her ikisi yanlış ise yanlış” kuralı bitsel için en az bir bit 1 ise üretilen bit 1 her iki bit de 0 ise üretilen bit 0 şeklindedir. Örneğin :

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val)

{

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main()

{

int val1=20,val2=10;

bitprint(val1);//0000 0000 0000 0000 0000 0000 0001 0100

printf("\n");

bitprint(val2);//0000 0000 0000 0000 0000 0000 0000 1010

printf("\n");

bitprint(val1 | val2);//0000 0000 0000 0000 0000 0000 0001 1110

}

Bitsel özel veya operatör (Bitwise exor operator)

Bu operatör aslında bitwise exclusive or ifadesinin kısaltmasıdır. Yaptığı iş ise iki sayının aynı bitleri ele alınırken aynı bitsel farklı ise 1 aynı ise 0 bitini üreten bir operatördür. Örneğin:

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val)

{

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main()

{

int val1=20,val2=10;

bitprint(val1);//0000 0000 0000 0000 0000 0000 0001 0100

printf("\n");

bitprint(val2);//0000 0000 0000 0000 0000 0000 0000 1010

printf("\n");

bitprint(val1 ^ val2);//0000 0000 0000 0000 0000 0000 0001 1110

}

Örnek Mülakat Sorusu : İki değişkeni geçici kullanmadan takasa eden programı yazınız.

#include <stdio.h>

int main() {

int a=30,b=20;

a^=b;

b^=a;

a^=b;

printf("a:%d b:%d",a,b);

}

Aynı soru macro haline getirildiği takdirde aşağıdaki gibi bir kod yazılabilir:

#include <stdio.h>

#define xor\_swap(a,b) ((a)^=(b),(b)^=(a),(a)^=(b));

int main()

{

int a=30,b=20;

xor\_swap(a,b);

printf("a:%d b:%d",a,b);

}

**Bitsel operatörlerin kullanılmasıyla yapılan tipik işlemler (Bitwise Manipulations)**

a-) Bit sayının n'inci bitini birlemek (to set the bit veya to turn the bit on)

Bunun en basit yolu 1'i n kadar sola kaydırıp söz konusu sayı ile or işlemine sokmaktır. Aşağıdaki gibi:

x |= (1 << n);

Test kodu da aşağıdaki gibi olacaktır:

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val){

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main(){

int val=20;

int n;

printf("Kacinci bit 1 yapilsin?");

scanf("%d",&n);//6 giriliyor

bitprint(val);//0000 0000 0000 0000 0000 0000 0001 0100

printf("\n");

val|=1<<n;

bitprint(val);//0000 0000 0000 0000 0000 0000 0101 0100

}

Bir sayının n. bitini 0 yapmak(to reset the bit, to turn the bit off, to clear the bit)

Bu işlem bir sayısını n kadar sola kaydırıp bitsel değilini aldıktan sonra sonra sayı ile ve işlemine sokularak yapılabilir. Örneğin :

val &= ~(1<<n);

Test kodu da aşağıdaki gibi olacaktır:

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val){

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main()

{

int val=20;

int n;

printf("Kacinci bit 0 yapilsin?");

scanf("%d",&n);//2 giriliyor

bitprint(val);//0000 0000 0000 0000 0000 0000 0001 0100

printf("\n");

val &= ~(1<<n);

bitprint(val);//0000 0000 0000 0000 0000 0000 0001 0000

}

Bir sayının n. bitini değiştirmek (to switch the bit veya to toggle the bit)

Bu işlem için 1 sayısını n kadar sola kaydırıp söz konusu sayı ile exor işlemine sokmak gerekir. Örneğin :

val ^= 1 << n;

Test kodu da aşağıdaki gibi olacaktır:

#include <stdio.h>

#include <stdlib.h>

void bitprint(int val){

static char buffer[200];

\_itoa(val,buffer,2);

printf("%032s\r",buffer);

}

int main(){

int val=20;

int n;

printf("Kacinci bit 0 yapilsin?");

scanf("%d",&n);//2 giriliyor

bitprint(val);//0000 0000 0000 0000 0000 0000 0001 0100

printf("\n");

val&=~(1<<n);

bitprint(val);//0000 0000 0000 0000 0000 0000 0001 0000

}

Örnek Soru : Bir tamsayının bitlerini bitsel operatörleri kullanarak yazdıran myBitPrint

isimli fonksiyon standart itoa fonksiyonunu kullanmadan yazınız.

#include <stdio.h>

void myBitPrint(int x){

for(int i=(int)(sizeof(x)\*CHAR\_BIT-1);i>=0;--i)

putchar((x>>i) & 1 ? '1':'0');

putchar('\n');

}

int main()

{

int x;

printf("Bir tamsayi giriniz?");

scanf("%d",&x);

myBitprint(x);

}

Yukarıdaki örnekte en baştaki bitin elde edilmesi için sayı bit cinsinden sizeof değerinin bir eksiği kadar sağa kaydırılır. Ardından bu işlem sırasıyla tüm bitlere yapılır. Bu bitler 1 ile ve işlemine sokulduğunda sonuç 1 ise bit 1 sonuç sıfır ise bit 0 olacaktır.

**Anahtar Notlar :** Bitsel ve operatörü kullanılarak bir sayının teklik çiftliği anlaşılabilir. Bu işlemi gerçekleştirmek için sayının 1 ile ve işlemine sokulması yeterlidir. Çünkü sayının içinde bir varsa sayı tek, sayının içinde bir yoksa sayı çift diyebiliriz. Örneğin :

#include <stdio.h>

int main()

{

int val;

printf("Bir sayi giriniz?");

scanf("%d",&val);

if(val & 1)

printf("Sayi tek");

else

printf("Sayi cift");

}

**Big Endian ve Little Endian Kavramları**

Bilgisayarların belleklerinde ve veri saklama düzenlerinde kullanılan iki önemli kavramdır: "big endian" ve "little endian". Bu terimler, çok sayıda veri işleme sistemlerinde verilerin nasıl saklandığını ifade ederler. İşte her iki kavramın açıklamaları:

1-) Big Endian: Big endian adı verilen bu saklama düzeninde, en yüksek sıradaki byte(most significant byte) en düşük bellek adresinde saklanır. Yani, verilerin en büyük parçası ilk bellek hücresinde bulunur. Örnek olarak, 0x11223344 değerini ele alalım. Big endian düzeninde bu değer şu şekilde saklanır:

Adres: 0 1 2 3

Değer: 11 22 33 44

Big endian mimarinde buradan da anlaşılabileceği gibi 11 değeri most significant byte olup en düşük bellek adresini kullanmaktadır. Bu mimari, özellikle IBM ve Motorola gibi bazı işlemci mimarilerinde kullanılmaktadır.

2-) Little Endian : Bu mimaride ise en büyük byte en büyük bellek adresinde tutulmaktadır. Yani, verilerin en büyük parçası son bellek bloğunda bulunmaktadır. Örneğin örnek değerimiz 0x11223344 olsun. Bu durumda değerler aşağıdaki gibi tutulur:

Adres : 1 2 3 4

Değer : 44 33 22 11

Bu durumda most significant byte 11’dir. Least significant byte ise 44’dür. Burda da görüldüğü gibi least signifacant byte en küçük bellek adresini kullanırken, most significant byte en büyük bellek adresini kullanmaktadır. Değerlerin adresleri bu şekilde dağıldığı sistemlere little endian sistem adı verilir. Bu mimari, x86 işlmecileri gibi birçok popüler işlemci mimarisinde kullanılmaktadır.

Örnek Soru : Bir sistemin big little endian mimarisine mi yoksa little endian mimarisine mi sahip olduğunu gösteren programı yazınız ve test ediniz.

Çözüm 1:

#include <stdio.h>

#include <inttypes.h>

int main(void)

{

uint32\_t x = 0x11223344;

uint8\_t\* ptr = (uint8\_t\*)&x;

if(\*ptr == 0x11)

{

printf("Big Endian\n");

}

else

{

printf("Little Endian\n");

}

}

Çözüm 2:

#include <stdio.h>

int main(void)

{

int x = 1;

if(\*(char\*)&x == 1)

{

printf("Little Endian\n");

}

else

{

printf("Big Endian\n");

}

}