**Bir değişkenin adresini bulmak**

* Her değişkenin hafızada bulundugu yeri gosteren özel bir adresi vardır. Bu adresi almak için ampersand (&) operatörü kullanılır.

long int j; // j , adresi 62FE4C olan long int bir degisken

ptr = &j; // ptr bu adresi saklayan bir degisken

#include <stdio.h>

void main() {

int j=1;

printf( "The value of j is: %d\n" , j );

printf( "The address of j is: %p\n" , &j );

//%p adresi portable ve hex olarak yazmayi saglar

}

The value of j is: 1

The address of j is: 000000000062FE4C

* & işareti solda kullanılamaz. Bir değişkenin adresi değiştirilemez.

&x = 1000; /\* ILLEGAL \*/

* ptr = &j; // ptr 'in tipi ne olmalıdır?

Adres tutmak için pointer adı verilen ozel bir tip kullanılır.

long \*ptr; /\* Long, ptr'nin gosterdiği adreste bulunan verinin tipini gosterir\*/

long long\_var;

ptr = &long\_var; /\*LEGAL\*/

long \*ptr;

float float\_var;

ptr = &float\_var; /\* ILLEGAL - ptr sadece bir long int'in adresini tutabilir \*/



#include <stdio.h>

int main () {

int j=1;

int \*pj;

pj = &j;

printf(“The value of j is: %d\n", j);

printf(“The adres of j is: %p\n", pj);

return 0;

}

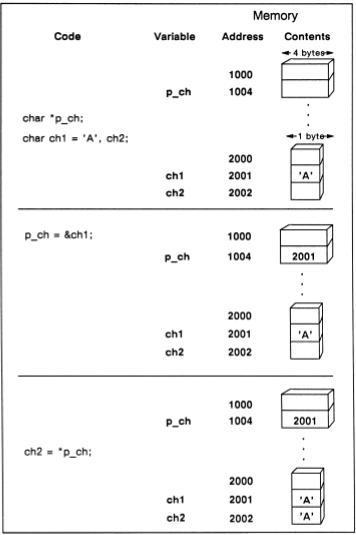
The result is:

The value of j is: 1

The address of j is: 000000000062FE4C

**Dereferencing**

* (\*) Operatörü hem pointer tanımlamak hem de o pointer'in gosterdigi adreste saklanan veriye ulasmak icin kullanır.



#include <stdio.h>

int main ()

{

char \*p\_ch;

char ch1 = 'A', ch2;

printf ( "The address of p\_ch is %p\n", &p\_ch );

p\_ch = &ch1;

printf ( "The value stored at p\_ch is %p\n", p\_ch );

printf( "The dereferenced value of p\_ch is %c\n" , \*p\_ch );

ch2 = \*p\_ch;

printf ( "The value stored at ch2 is %c\n", ch2 );

return 0;

}

The address of p\_ch is 000000000062FE40

The value stored at p\_ch is 000000000062FE3F

The dereferenced value of p\_ch is A

The value stored at ch2 is A

* Pointer initialization:

int j; int j;

int \*ptr\_to\_j = &j; //LEGAL int \*ptr\_to\_j = &j; //ILLEGAL

**Pointer Arithmetic**

* C dili pointer ile integer’lari isleme sokmaya izin verir. p bir pointer olmak uzere,

p+3 legal bir işlemdir.

* p bir adres olduğu için p ile yapılan aritmetik işlem de adres üretir. Ancak basitçe p’nin değerine 3 eklemektense SCALING yapılır. p hangi tipte verinin adresini tutarsa tutsun, p+3 p’den 3 nesne sonrayı işeret eder

char \*p;

char c='A';

p=&c; //p=1000 olsun

p=p+3; // p = 1001 olur

* 32 bitlik bir sistemde p int\* olarak tanımlansaydı;

p=p+3 //p= 1000+ (4\*3) = 100C

* &a[3] - &a[0] -> 3 döndürür

&a[0] - &a[3] -> -3 döndürür

**Dizi elemanlarına pointer uzerinden ulasma**

* short ar[4];

short \*p;

p = &ar[0] ;

\*(p+3) // ar[3]

* Dizi isimleri tek başlarına kullanıldıklarında pointer olarak işleme alınırlar

ar <-> &ar[0]

ar[n] <-> \*(ar + n)

#include <stdio.h>

int main()

{

int a[5]={1,2,3,4,5};

int \*p1, \*p2;

int j;

char \*p3;

p1=a;

p2 = p1 + 4; /\* legal \*/

j = p1 - p2; /\* legal j is assigned -4 \*/

printf("j:%d\n",j);

p1 = p2 - 2; /\* legal compatible pointer types \*/

printf("\*p1:%d\n",\*p1);

//j = p1 - p3; /\* ILLEGAL -- different pointer types\*/

return 0;

}

j:-4

\*p1:3

**Fonksiyonlar**

* Fonksiyonlar C dilinin temelidir ve tüm işlemin gerçekleştiği yerlerdir.
* Her fonksiyon, kendi başına bir kod bloğudur. Bu yüzden fonksiyonlar kendi tanım alanlarını oluştururlar.
* Fonksiyon bloğunun yapısı;

dönüş\_tipi fonksiyon\_adi(parametre listesi)

{

işlem bloğu

}

* dönüş\_tipi, fonksiyon sonlandiginda fonksiyonun çağırıldığı noktaya gidecek olan değerin tipini belirtir.
* Fonksiyonlar aynı anda bir deger dondurebilir. Deger dondurmeyebilir (void)
* Parametre listesi, fonksiyon çağırılırken kullanılması gereken değişkenler ve onların veri tiplerinden olusur. Fonksiyon parametre kullanmadan da çağırılabilir.
* Fonksiyon bir programda üç şekilde görünebilir: (Prototip tanımı, Fonksiyon gövdesi ve Fonksiyonun çağırılması)
* C’de fonksiyona parametre gönderme işlemi 2 ayrı şekilde yapılır: Değer ile ve referans ile çağırma.
* Fonksiyona gönderilen parametrelerde fonksiyon tarafından yapılan değişikliklerin ana fonksiyona yansımasını sağlamak için refernas ile çağırma yapılır.
* Toplama, çarpma, kare alma ve swap işlemleri yapan fonksiyonlar yazalım;

#include <stdio.h>

void swap(int\*,int\*); // Prototip tanımları

void yaz(void);

int topla (int, int);

void carp (int, int , int \*);

int kareAl(int);

int main()

{

int i, j, result;

i = 10;

j = 20;

result=topla(i,j); // Deger ile çağırma

printf("topla: %d\n",result);

carp(i,j,&result);

printf("carp: %d\n",result);

result=kareAl(i,i);

printf("kare: %d\n",result);

printf("Yer degistirme oncesi i ve j: %d %d\n", i, j);

swap(&i, &j); //Referans ile çağırma

printf("Yer degistirme sonrasi i ve j: %d %d\n", i, j);

yaz();

}

void swap(int \*x, int \*y)

{

int temp;

temp = \*x;

\*x = \*y;

\*y = temp;

}

void yaz(void)

{

printf("yaz\n");

}

int topla (int sayi1, int sayi2)

{

int toplam;

toplam=sayi1+sayi2;

return toplam;

}

void carp (int sayi1, int sayi2, int \*carpim)

{

\*carpim=sayi1\*sayi2;

//carpimi return ile dondurmek yerine adresini gonderdik

}

int kareAl(int i)

{

int kuvvet;

carp(i,i,&kuvvet); //bir fonksiyon baska bir fonksiyonu cagirabilir

return kuvvet;

}

**Fonksiyonlarda Diziler**

* Verilen bir dizinin elemanlarını toplayan fonksiyonlar yazalım;

#include <stdio.h>

int toplaDiziv1(int \*,int);

int toplaDiziv2(int \*,int);

int main()

{

int dizi[20],n,i,toplam;

int \*dizi2;

printf("Dizinin eleman sayisiniz giriniz\n");

scanf("%d",&n);

printf("elemanlari giriniz\n");

for (i=0;i<n;i++)

scanf("%d",&dizi[i]);

toplam=toplaDiziv1(dizi,n);

printf("toplam v1:%d\n",toplam);

toplam=toplaDiziv2(dizi,n);

printf("toplam v2:%d\n",toplam);

return 0;

}

int toplaDiziv1(int \*dizi,int boyut)

{

int i=0,toplam=0;

for (i=0;i<boyut;i++)

toplam+=dizi[i]; //\*(dizi+i) seklinde de yazilabilir

return toplam;

}

int toplaDiziv2(int dizi[],int boyut)

{

int i=0,toplam=0;

for (i=0;i<boyut;i++)

toplam+=\*(dizi+i); //dizi[i] seklinde de yazilabilir

return toplam;

}

**Fonksiyonlarda Matrisler**

* Verilen bir matrisin elemanlarını toplayan bir fonksiyon yazalım;

#include <stdio.h>

int toplaMat (int [][50], int, int);

int main()

{

int mat[50][50],i,j,n,m;

printf("matrisin satir ve sutun sayisini veriniz\n");

scanf("%d %d",&n,&m);

printf("elemanlari veriniz\n");

for (i=0;i<n;i++)

for (j=0;j<m;j++)

scanf("%d",&mat[i][j]);

printf("toplam:%d\n",toplaMat(mat,n,m));

}

int toplaMat (int mat[][50], int dim1, int dim2)

{

int topla=0,i,j;

for (i=0;i<dim1;i++)

for (j=0;j<dim2;j++)

topla=topla+mat[i][j];

return topla;

}