***PROFESYONEL C PROGRAMLAMA DERS NOTLARI***

***Formatlı/formatsız giriş çıkış işlemi:***

Formatlı: insanın anlayacağı şekilde

Formatsız: bilgisayarın anlayacağı şekilde

Ekrana hexadecimal yazdırmak için printf(“%x”) kullanılır.

Ekrana octal yazdırmak için printf(“%o”) kullanılır.

Floatlarda virgülden sonra kaç basamak yazacağını belirtmek için:

printf(“%**.**?f”) yap. ? istediğin basamak sayısını belirtiyor.

Sayıları istediğin kadar alana yazdırmak için:

printf(“%?d”) yap.

1234

234

34

4

gibi.

Herhangi bir örnek:

**int** a=1234, b=234, c=34, d=4;

printf("a=%4d\n",a);

printf("b=%4d\n",b);

printf("c=%4d\n",c);

printf("d=%4d\n",d);

printf("%d+%d+%d+%d=%d",a,b,c,d,a+b+c+d);

**a=1234**

**b= 234**

**c= 34**

**d= 4**

**1234+234+34+4=1506**

Printf in geri dönüşü int türdendir.

Yani içindeki karakter sayısını yazdırıyor.

**int** x=printf("ali");

printf(" x=%d\n",x);

**ali x=3**

***scanf hakkında notlar:***

scanf in format uyumsuzluğunda yaptıklarına dair bir örnek:

**int** x,y;

printf("enter two integer numbers:\n");

scanf("%d",&x);

printf("%d\n",x);

scanf("%d",&y);

printf("%d\n",y);

**enter two integer numbers:**

123asude

**123**

**0**

**Program ended with exit code: 0**

Y yi almamıza izin vermedi bile, kendi kendine bir değeri ekrana yazdırdı.

Scanfte int alırken %d, float alırken %f, double alırken %lf (kucuk L)

ama double ı printfte %f ile yazdırabilirsin sıkıntı olmaz.

Float ile double farkı şu:

**float**” 6 basamaklı değerleri desteklerken, “**double**” 14 basamaklı değerleri desteklemektedir. Yani projenizde “**float**” veya “**double**” değeri kullanırken değişkenin alabileceği değerin basamak sayısını göz önünde tutabilirsiniz.

Scanfin geri dönüş değeri başarılı olduğu set edebildiği değer sayısıdır:

**int** x,y,z;

printf("Enter three numbers:\n ");

**int** a=scanf("%d%d%d",&x,&y,&z);

printf("%d %d %d\n",x,y,z);

printf("scanf in donus degeri:%d\n",a);

**Enter three numbers:**

12 13 14

**12 13 14**

**scanf in donus degeri:3**

**Enter three numbers:**

12 13 asude

**12 13 0**

**scanf in donus degeri:2**

***getchar putchar***

tek bir karakter!

printf("Enter anything: \n");

**int** ch=getchar();

printf("%d\n",ch);//karakterin ascideki degeri

printf("%c\n",ch);//karakterin goruntusu

**int** ch2=getchar();

printf("%d\n",ch2);//karakterin ascideki degeri

printf("%c\n",ch2);//karakterin goruntusu

**Enter anything:**

AB

**65**

**A**

**66**

**B**

Kullanım yerine bir örnek:

printf("for YES please enter (y), for NO please enter (n):\n");

**int** c=getchar();

**if** (c=='y') {

printf("You said YES");

}

**else** **if**(c=='n'){

printf("You said NO");

}

**else**{

printf("invalid choice");

}

getcharın geri dönüş değeri aldığı karakterin kod numarasıdır.(ASC2 tablosuna göre)

printf("Herhangi bir sey yaziniz:\n");

**int** x=getchar();

printf("%d\n",x);

**Herhangi bir sey yaziniz:**

A

**65**

Putchar

Girilen karakterin kodunu ekrana yazdırır.(Yine asciye göre.)

Yani şöyle, ekrana A yazdırmak istiyorsan içeriye 65 demelisin.

Ya da bir değişkene (integer bir değişken) 65 atarsın putcharın içine o değişkeni koyarsın.

**int** x=65;

putchar(x);

**A**

Tek tırnak içinde A yazarsan da yine A yı ekrana yazdırabilirsin.

Şimdi ascideki karakterleri topluca yazdıralım:

New line’ın ascideki değerini bilmiyorsan tek tırnak içinde yazarak sorunu çözebilirsin.

**int** i;

**for** (i=32; i<126; i++) {

printf("%d:",i);

putchar(i);

putchar('\n');

}

Ayrica printf ile de şöyle bir şey yapabilirsin. (Ben ilk gördüğümde şaşırdım.)

**int** i;

**for** (i=32; i<126; i++) {

printf("%d:%c\n",i,i);

}

**32:**

**33:!**

**34:"**

**35:#**

**36:$**

**37:%**

**38:&**

**39:'**

**40:(**

**41:)**

**42:\***

**43:+**

**44:,**

**45:-**

**46:.**

**47:/**

**48:0**

**49:1**

**50:2**

**51:3**

**52:4**

**53:5**

**54:6**

**55:7**

**56:8**

**57:9**

**58::**

**59:;**

**60:<**

**61:=**

**62:>**

**63:?**

**64:@**

**65:A**

**66:B**

**67:C**

**68:D**

**69:E**

**70:F**

**71:G**

**72:H**

**73:I**

**74:J**

**75:K**

**76:L**

**77:M**

**78:N**

**79:O**

**80:P**

**81:Q**

**82:R**

**83:S**

**84:T**

**85:U**

**86:V**

**87:W**

**88:X**

**89:Y**

**90:Z**

**91:[**

**92:\**

**93:]**

**94:^**

**95:\_**

**96:`**

**97:a**

**98:b**

**99:c**

**100:d**

**101:e**

**102:f**

**103:g**

**104:h**

**105:i**

**106:j**

**107:k**

**108:l**

**109:m**

**110:n**

**111:o**

**112:p**

**113:q**

**114:r**

**115:s**

**116:t**

**117:u**

**118:v**

**119:w**

**120:x**

**121:y**

**122:z**

**123:{**

**124:|**

**125:}**

***if ve switch***

Basit bir if kullannım örneği ve öncesinde ufak bir fonksiyon:

#include <stdio.h>

**int** max\_of\_value(**int** x,**int** y ){

**if** (x>y) {

**return** x;

}

**else**

**return** y;

}

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** num1,num2;

printf("Please enter two values that are not same: \n");

scanf("%d%d",&num1,&num2);

printf("Max value is: %d\n",max\_of\_value(num1, num2));

**return** 0;

}

Bir if in içine blok açmadan birden fazla statement virgüller sayesinde konabilir:

printf("Lutfen bir x degeri giriniz:");

scanf("%d",&x);

printf("Lutfen bir y degeri giriniz:");

scanf("%d",&y);

**if** (x>y)

x++,y++,printf("Yeni x ve y degerleriniz:%d ve %d",x,y);

**else**

x--,y--,printf("Yeni x ve y degerleriniz:%d ve %d",x,y);

Eğer if in doğru kısmına hiçbir şey koymak istemiyorsan (çok tercih edilen bir şey olmasa da)

Statement’ı boş bırakıp sonuna ; koyabilirsin. Sistem hata vermez.

***Dangling else (yani kimin eli kimin cebinde)***

Hangi else kimin if’i belli olmadığı durumlar için söylenen bir tabirdir.

***SWITCH***

Bir ifadenin farklı farklı değerleri için programın akışını farklı farklı noktalara yönlendirmek.

Uyarı:

case x gibi bir kullanım olamaz, case 1, case 2 gibi belirtilmelidir.

default olmak zorunda değildir, ancak kullanılırsa şu anlama gelir: Program case 1,2,...,5 olmuyorsa default olanı yapar.

Kullanımı:

switch(integer expression){

case constant expression :

case 1:

case 2:

case 3:

default :

}

örnek:

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** not;

printf("Lutfen ders notunuzu giriniz:\n");

scanf("%d",&not);

**switch** (not) {

**case** 1:

printf("Yetersiz\n");

**break**;

**case** 2:

printf("Gecerli\n");

**break**;

**case** 3:

printf("Orta\n");

**break**;

**case** 4:

printf("İyi\n");

**break**;

**case** 5:

printf("Cok iyi\n");

**break**;

**default**:

printf("Gecersiz bir not girdiniz!");

**break**;

}

**return** 0;

}

örnek 2:

**int** not;

printf("Lutfen ders notunuzu giriniz:\n");

scanf("%d",&not);

**switch** (not) {

**case** 5:

**case** 4:

**case** 3:

**case** 2:

printf("Gectiniz.\n");

**break**;

**case** 1:

**case** 0:

printf("Kaldiniz.\n");

**break**;

**default**:

printf("Gecersiz bir not girdiniz!");

**break**;

}

**int** month;

printf("Please enter the month that you want.(like this: 1,2,3,..,12)This program will be find which season include it:\n");

scanf("%d",&month);

**switch** (month) {

**case** 12:

**case** 1:

**case** 2:

printf("Your month is in winter.\n");

**break**;

**case** 3:

**case** 4:

**case** 5:

printf("Your month is in spring.\n");

**break**;

**case** 6:

**case** 7:

**case** 8:

printf("Your month is in summer.\n");

**break**;

**case** 9:

**case** 10:

**case** 11:printf("Your month is in fall.\n");

**break**;

}

switch ile ilgili güzel bir örnek:

//target:turn this date (15.05.2000) to this version(15th day of May 2000)

**int** day,month,year;

printf("Enter the date like this please(dd/mm/yyyy): ");

scanf("%d%d%d",&day,&month,&year);

printf("This is %d",day);

**switch** (day) {

**case** 1:**case** 11:**case** 21:**case** 31:

printf("st");

**break**;

**case** 2:**case** 12:**case** 22:

printf("nd");

**break**;

**case** 3:**case** 13:**case** 23:

printf("rd");

**break**;

**case** 4:**case** 14:**case** 24:

printf("th");

**break**;

**default**:

printf("th");

}

printf(" Day of ");

**switch** (month) {

**case** 1:printf("January");**break**;

**case** 2:printf("February");**break**;

**case** 3:printf("March");**break**;

**case** 4:printf("April");**break**;

**case** 5:printf("May");**break**;

**case** 6:printf("June");**break**;

**case** 7:printf("July");**break**;

**case** 8:printf("August");**break**;

**case** 9:printf("September");**break**;

**case** 10:printf("October");**break**;

**case** 11:printf("November");**break**;

**case** 12:printf("December");**break**;

}

printf(" %d \n",year);

**Enter the date like this please(dd/mm/yyyy):** 15 5 2000

**This is 15th Day of May 2000**

if için başka bir örnek:

**int** hour,min;

printf("Enter a time according to 24 hours:");

scanf("%d %d",&hour,&min);

**if** (hour<12) {

printf("%.2d.%.2d am\n",hour,min);

}

**if** (hour==12) {

printf("%.2d.%.2d pm\n",hour,min);

}

**else** **if**(hour>=12)

printf("%.2d.%.2d pm\n",hour%12,min);

Örnek: (Videoda if kullandı ama ben switch ile yapmayı tercih ettim)

**int** opr;

**float** n1,n2,result;

printf("Enter two numbers please:\n");

scanf("%f %f",&n1,&n2);

printf("Which operation so you want?\n For addition(1),for subtraction (2), for multiplication (3), for division(4):\n");

scanf("%d",&opr);

**switch** (opr) {

**case** 1:

result=n1+n2;

printf("Outcome is: %.2f\n",result);

**break**;

**case** 2:

result=n1-n2;

printf("Outcome is: %.2f\n",result);

**break**;

**case** 3:

result=n1\*n2;

printf("Outcome is: %.2f\n",result);

**break**;

**case** 4:

**if** (n2==0) {

printf("Error!");

}

**else**{

result=n1/n2;

printf("Outcome is: %.2f\n",result);

**break**;

}

}

**Enter two numbers please:**

10

4

**Which operation so you want?**

**For addition(1),for subtraction (2), for multiplication (3), for division(4):**

4

**Outcome is: 2.50**

***If Switch başlığı altındaki son örnek.***

***2 önemi var bu kodun.***

***İlki: scanf ile karakter alırkenki sıkıntıdan ve çözümünden bahsettik.***

***İkincisi de kodun devam etmesini istemediğimizde nasıl return 0 ile çıkılabileceğini öğrendik.***

***(Ayrıca hem üstteki hem de alttaki kodda göreceksin ki switch deyimi içinde if kullanabiliyoruz.)***

#include <stdio.h>

**int** main(**int** argc, **const** **char** \* argv[]) {

**char** opr;

**float** n1,n2,result;

printf("Enter two numbers please:\n");

scanf("%f %f",&n1,&n2);

printf("Please choose the operator that you want:('+','-','\*','/')\n");

scanf("%c",&opr);

scanf("%c",&opr);//char her karakteri kabul ettiginden ustte floattan sonraki enterı almıs oldu, bunu onlemek icin de tekrar %c ile opr aldım ki sorun olusmasin. cop degerle de yapabilirsin.

//ya da char ile alma islemini en basa alırsın boylece entera basmamıs olursun oncesinde ve hata vermez.

**switch** (opr) {

**case** '+':

result=n1+n2;

printf("%.2f",result);

**break**;

**case** '-':

result=n1-n2;

printf("%.2f",result);

**break**;

**case** '\*':

result=n1\*n2;

printf("%.2f",result);

**break**;

**case** '/':

**if** (n2==0) {

printf("Error.");

**return** 0;

}

result=n1/n2;

printf("%.2f",result);

**break**;

**default**:

**break**;

}

**return** 0;

}

**Enter two numbers please:**

10

0

**Please choose the operator that you want:('+','-','\*','/')**

/

**Error.Program ended with exit code: 1**

***Operatorler***

***Türlerine göre:***

Aritmetik Operatörler

Atama Operatörleri

Karşılaştırma Operatörleri

Mantıksal Operatörler

Bitsel Tabanlı Operatörler

Koşul Operatörü

Pointer Operatörleri

sizeof operatörü

Artırma-Azaltma Operatörleri

Tür Dönüştürme Operatörleri

***Operant sayısına göre:***

Unary

Binary

Ternary

***Konumuna göre:***

Postfix operartör (Operanttan sonra geliyorsa)

Prefix operatör (Operanttan önce geliyorsa)

Infix operatör (Ortada olan)

örnek:

x++ unary postfix operatör

!x unary prefix operatör

x+y binary infix opeatör

1. ***Aritmetik Operatörler***

addition(+)

subtraction(-)

multiplication(\*)

division(/)

remaninder(%)(operantlar tam sayı olmak zorunda)

***tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu***

1. ***Atama operatörleri***

b.1 Basit Atama Operatörü

NOT:

Side effect: operatör operandını değiştiriyorsa buna yan etkili operatör denir.Mesela atama operatörlerinde side effect vardır.

Atamanın yönleriyle ilgili önemli bir örnek (Mulakat sorusu):

**int** x;

**float** y;

y=x=6.24;

printf("x: %d y: %f",x,y);

**x: 6 y: 6.000000**

b.2 Birleşik Atama Operatörü (İşlemli Atama Operatörü)

+=

-=

\*=

/=

%=

şu şekilde kullanılır:

**int** i=5;

i=i+3;

printf("%d",i);

**int** i=5;

i+=3;

printf("%d",i);

yani i=i+3 ile i+=3 aynı şeydir, her ikisinde de sonuç 8 olacaktır.

(-,\*,/,% da aynı şekilde)

1. ***Artırma ve azaltma operatörleri***

++i ile i++ ya da --i ile i—farklı şeylerdir. Aslında ikisi de 1 artırma ya da azaltma operatörüdür ancak farkları şöyledir.

++i yaptığımızda i değerine DİREKT 1 ekler.

i++ yaptığımızda ise i değerine belli olmayan bir süre sonra 1 ekler.(Genelde 1 alt satırda itibarendir.)

Şimdi örnekle açıklayalım.

**int** i=10;

**int** j=10;

printf("i=%d\n",++i);

printf("i=%d\n",i);

printf("i=%d\n",j++);

printf("i=%d\n",j);

**i=11** Görmüş olduğumuz gibi i yi direkt 11 yaptı, j yi ise 1 alt satırda yazdırdığımızda.

**i=11**

**i=10**

**i=11**

Örnek:

**int** i=10;

**int** j=20;

**int** k= ++i + j++;

printf("k=%d\n",k);

Böyle bir örnekte k 31 olarak ekrana yazdırılacaktır.

***d)Karşılaştırma Operatörleri***

>

<

>=

<=

==

!=

Bu ifadeler lojik yorumlamaya tabiidir. Yani doğruysa 1 yanlışsa 0 şeklinde.

**int** x=10>9;

printf("%d\n",x);

**int** y=10>11;

printf("%d\n",y);

**1** Program, birinci ifade doğru olduğu için 1, ikinci ifade yanlış olduğundan 0 yazdı.

**0**

Ek not: if statement da lojik yorumlamaya tabiidir. zero ya da non zero şeklinde çalışma prensibi vardır.

***e)Lojik operatörler***

! değil operatörü

&& ve operatörü

|| veya operatörü

!op1 op1 non zero değer ise sonuç falsedur.

!op1 op1 zero ise sonuç truedur.

Örnek:

**int** x;

printf("Bir sayi giriniz:");

scanf("%d",&x);

printf("!x=%d\n",!x);

Beklediğimiz şey:

x e 0 hariç bir şey girersem değilini aldığımız için program 0 yazdıracaktır.

Ancak 0 girersem yine değilini aldığımdan program 1 yazdıracaktır.

**Bir sayi giriniz:**15

**!x=0**

**Bir sayi giriniz:**0

**!x=1** şeklinde.

Ek Bilgi:

if in içine istediğimiz ifadenin direkt kendisini değil de değilini yazabiliyorsak onu yazarız, tekrar değilini aldığımızda istediğimiz şeyi elde etmiş oluruz.

Ve Veya kısmında ek bir bilgi olmadığı için not almadım.

Örnek 1:

//50-20-10-5-1 seklinde banknotlar gecerli

**int** para;

**int** ellilik, yirmilik,onluk,beslik,birlik;

printf("Lutfen banknotlar halinde hesaplamak istediginiz miktarı giriniz:");

scanf("%d",&para);

ellilik=para/50;

para=para%50;

yirmilik=para/20;

para=para%20;

onluk=para/10;

para=para%10;

beslik=para/5;

para=para%5;

birlik=para/1;

printf("Paraniz:%d adet ellilik, %d adet yirmilik, %d adet onluk, %d adet beslik, %d adet birlikten olusmaktadir.\n",ellilik,yirmilik,onluk,beslik,birlik);

**Lutfen banknotlar halinde hesaplamak istediginiz miktarı giriniz:**98

**Paraniz:1 adet ellilik, 2 adet yirmilik, 0 adet onluk, 1 adet beslik, 3 adet birlikten olusmaktadir.**

Örnek 2: 11lik sayı tabanına dönüştürmeyle alakalıydı ona sonra bakarsın istersen.

***Sequence Point***

Yan etki noktası öyle bir noktadır ki ondan önceki tüm kodlar tamamlanmış olmalıdır.

Mesela:

int x=5, y=x , z=3\*y;

yaptığımızda virgüller yan etki noktasıdır ve ondan önceki kod tamamlanmış olmalıdır.

Zaten böyle yapmazsak sıkıntı çıkacaktı.

Mesela:

Şu durumda yan etki noktası kullanılmadığından tanımışız davranış olur.

**int** i=5;

i=i++;

Ayrıca aşağıdakiler de yan etki noktalarıdır:

* &&
* ||
* ?:
* return statement
* (if- switch- while- do while) ın parantez içleri

örnek:

**int** x=0;

**int** z= x++||x;

printf("%d",z);

//burada olan şu, || yine sequence noktamız ve ondan önceki kod tamamlanmış olmalı.

//x++ dediğimizde 1 olacak ancak direkt olmuyor o yüzden veyanın sol tarafı hala 0

// sağ tarafında ise x artık 1 ve veya da 0 ile 1, 1 sonucunu veriyordu.

(Yani o kodun tamamlanıyor ancak veyadan sonra gösteriyor kendini.)

**1**

Sol tarafın direkt 1 olmasını istesek ne yapardık x++ değil ++x kullanırdık.

***Virgul Operatörü***

Virgül operatörü aşağıdaki gibi kullanıldığında sağdaki operandı alacaktır.

**int** x=20;

**int** y=10;

**int** z=(x,y);

printf("%d",z);

**10**

Virgül operartörü

* if bloğu kullanmak istemediğimizde birkaç şey ifin içinde yaptırabilmemizi de sağlar.
* bir değişkeni for içine alabilmemizi ve böylece döngüden çıktığımızda yine aynı değişkeni kullanmaya devam edebilmemizi sağlar.

**for** (**int** i=1,sum=0; i<5; i++) {

printf("%d\n",sum+=i);

}

**int** sum=25;

printf("%d",sum);

**1**

**3**

**6**

**10**

**25**

1. for içinde şartlı bir kullanım da yapabilmemizi sağlar.

**for** (**int** i=1,k=1; i<50,k<40; i++,k\*=3) {

printf("%d %d\n",i,k);

}

**1 1**

**2 3**

**3 9**

**4 27**

şeklinde.

***Koşul Operatörü (Ternary Operatör)***

C de 3 operandı olan tek operatördür.

?: sembollerinden oluşur.

**expr1**?**expr**2:**epr3**

expr 1 lojik yorumlamaya tabiidir. zero ya da non zero şeklinde. eğer true değer üretiyorsa expression 2, false değer üretiyorsa expression 3 yapılır.

mutlak değer bulan bir koşul operatörü örneği:

printf("Please enter a number: ");

scanf("%d",&x);

result = (x>0?x:-x);

printf("%d",result);

***sizeof operatörü***

sizeof operatörü operandı olan ifadenin compile timeda sistemde kaç byte yer kapladığını bulur.

2. öncelik seviyesindedir ve yönü sağda sola doğru olan bir operatördür.

sizeof için %zu formatlayıcısı kullanılır.

printf("sizeof(char)= %zu\n",**sizeof**(**char**));

printf("sizeof(signed char)= %zu\n",**sizeof**(**signed** **char**));

printf("sizeof(unsigned char)= %zu\n",**sizeof**(**unsigned** **char**));

printf("sizeof(int)= %zu\n",**sizeof**(**int**));

printf("sizeof(float)= %zu\n",**sizeof**(**float**));

printf("sizeof(double)= %zu\n",**sizeof**(**double**));

printf("sizeof(long double)= %zu\n",**sizeof**(**long** **double**));

printf("sizeof(long)= %zu\n",**sizeof**(**long**));

printf("sizeof(long long)= %zu\n",**sizeof**(**long** **long**));

**sizeof(char)= 1**

**sizeof(signed char)= 1**

**sizeof(unsigned char)= 1**

**sizeof(int)= 4**

**sizeof(float)= 4**

**sizeof(double)= 8**

**sizeof(long double)= 16**

**sizeof(long)= 8**

**sizeof(long long)= 8**

Ayrıca sizeof operatörü

1. dinamik bellek yönetimi fonksyionlarında
2. dizi boyutunu hesaplamada

da kullanılır.

Bir mülakat sorusu (fosforlu yer)

**int** a;

printf("%zu\n",**sizeof**(**sizeof**(15)+**sizeof**(9.5)));

printf("%zu\n",**sizeof**(a));

**8**

**4**

**SOR: benim derleyicimde int 4 olmasına rağmen sorunun cevabı 8 oluyor neden?**

***LOOPS***

***while loop***

while (control expression)

loop body

Control expression yine lojik yorumlamaya tabiidir.(non zero ise true, zero ise false şeklinde)

***do while deyimi***

kod en az 1 kez yürütülür ardından kontrol yapılır.syntaxı:

do{

islem

} while(expression);

ifade doğruysa tekrar do’ya gidecek yanlışsa whileden sonraki işleme geçecek.

örnek1:

**int** i=5;

**do** {

printf("%d\n",i);

i--;

} **while** (i>0);

**5**

**4**

**3**

**2**

**1**

örnek2:

**int** i=5;

**do** {

printf("%d\n",i);

i--;

} **while** (i>6);

**5**

örnek 3:

#include <stdio.h>

//kullanicinin girdigi sayinin basamak sayisini bulan kodu do while dongusu yardımıyla yazalim

**int** ndigit(**int** val){

**int** nd=0;

**do** {

val=val/10;

nd++;

} **while** (val>0);

**return** nd;

}

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** num;

printf("Please enter a nonnegative number to count it's digit number:");

scanf("%d",&num);

printf("%d\n",ndigit(num));

**return** 0;

}

do while kullanmamız bize kullanıcı 0 girse bile basamak sayısının 1 olarak hesaplanmasını sağladı.

whilesız da şöyle yapardık:

#include <stdio.h>

//kullanicinin girdigi sayinin basamak sayisini bulan kodu do while dongusu yardımıyla yazalim

**int** ndigit(**int** val){

**int** nd=0;

**if** (val==0) {

**return** 1;

}

**while** (val>0) {

val=val/10; //bunu val/=10 seklinde de yazabilirsin unutma

nd++;

}

**return** nd;

}

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** num;

printf("Please enter a nonnegative number to count it's digit number:");

scanf("%d",&num);

printf("%d\n",ndigit(num));

**return** 0;

}

***for deyimi:***

biliyorsun ek bir şey demedi.

***nested loops (iç içe döngüler):***

örnek:

bu kodda amaclanan goruntu:

\* i=1 j=1

\*\* i=2 j=1 i=2 j=2

\*\*\* i=3 j=1 i=3 j=2 i=3 j=3

\*\*\*\* i=4 j=1 i=4 j=2 i=4 j=3 i=4 j=4

\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** i,j;

**for** (i=1; i<10; i++) {

**for** (j=1; j<=i; j++) {

printf("\*");

}

printf("\n");

}

**return** 0;

}

***Bir döngüden çıkış yolları:***

***1)Kontrol ifadesinin yanlış olması ile***

***2)break statement***

***3)return statement***

***4)go to statement***

***5)exit abort fonksiyonları***

***Break statement***

**int** i=20;

**while** (i>0) {

printf("%d ",i);

**if** (i%7==0) {

**break**;

}

i--;

}

**20 19 18 17 16 15 14**

Örnek:

**int** number,flag;

printf("Enter a number please: \n");

scanf("%d",&number);

**int** i;

**for** (i=2; i<number; i++) {

**if** (number%i==0) {

flag=1;

**break**;

}

}

**if** (flag) {

printf("The number you enter is not prime\n");

}

**else**

printf("The number you enter is prime\n");

***goto statement:***

**goto** Git;

printf("program buraya girmeyecektir.");

Git:

printf("Program burdan devam edecektir.");

**Program burdan devam edecektir.**

Git:

printf("Program burdan devam edecektir.");

**goto** Git;

böyle yapsaydık sonsuz döngü oluşurdu.

*Çıkmamız gereken tek bir yer varsa break kullanmak daha mantıklıdır. Ancak içiçe birkaç döngüden toptan çıkmak istiyorsak goto deyimini kullanmamız gerekir.*

***TYPE CONVERTION (TÜR DÖNÜŞTÜRME)***

iki tür tür dönüşümü vardır:

Implicit type convertion (üstü kapalı, gizli): derleyici kendisi yapar.

explicit type convertion (açık): bunda ise type cast operatörü kullanarak biz yaparız.

***Implicit type convertion:***

*Aşağıdaki durumlarda derleyici tür dönüşümü yapar:*

1. Aritmetik veya lojik ifadeler arasındaki tür dönüşümlerinde
2. Fonksiyonun parametresiyle argümanları eşleşmiyorsa
3. Fonksiyonun geri dönüş değeri türü return deyimiyle eşleşmiyorsa
4. Atama işleminde sağ taraftaki değerle sol taraftaki değer örtüşmüyorsa

int altı türler arasında yapılan dönüşümlere integral promotion denir.

Mesela 2.ye örnek verelim:

#include <stdio.h>

**void** deneme(**double** x){

}

**int** main(**int** argc, **const** **char** \* argv[]) {

deneme(12.2);

**return** 0;

}

oluşturduğumuz fonksiyonun içine double yazmamıza ve çağırdığımız fonksiyona bir float değer yazmamıza rağmen compiler bunu otomatik düzenledi.

3.ye örnek:

#include <stdio.h>

**double** deneme(**int** x){

**float** y=30.5;

**return** y;

}

4.ye örnek:

**int** x=20;

**float** y=30.343;

y=x;

printf("%f",y);

***Type Cast Operatörü***

**int** ival1=10;

**int** ival2=4;

**double** result=ival1/ival2;

printf("%f\n",result);

//bu durumda her iki degisken int oldugundan sonuc kusuratı kaybetmis olduk

**2.000000**

Aynı kodu şöyle yazarsak sonuç istediğimiz gibi olacaktır:

**int** ival1=10;

**int** ival2=4;

**double** result=(**double**)ival1/ival2;

printf("%f\n",result);

**2.500000**

Ancak aynı kodu şöyle yazarsak önce parantez içi yaptığından yine küsüratı kaybedeceğiz ve type cast yapmamıza rağmen 2.000000 degerini alacağız.

**int** ival1=10;

**int** ival2=4;

**double** result=(**double**)(ival1/ival2);

printf("%f\n",result);

**2.000000**

Önemli bir mülakat sorusu: Ekranda ne yazar?

**unsigned** **int** y=1;

**int** x=-1;

**if** (y>x) {

printf("True");

}

**else**

printf("False");

Ekranda False yazacaktır:

Çünkü y değerinin karşılığı 0000 0000 0000 0000 0000 0000 0000 0001 iken (int ler 4 byte idi unutma)

x değeri işaretli olduğundan 1111 1111 1111 1111 1111 1111 1111 1111 olacaktır.

***FONKSİYONLAR***

Belirli bir işi yerine getiren alt programlara fonksiyon denir.

Fonksiyon şöyle yazılır:

geri dönüş değerinin türü fonksiyonun adı (girdi1 türü ve adı, girdi2 türü ve adı,... ){

işlemler;

return ? ;

}

fonksiyonun parametresidir.

fonksiyonun bodysi olarak geçer.

Basit bir örnekle başlayalım:

#include <stdio.h>

**int** toplam(**int** n1,**int** n2){

**int** toplam=n1+n2;

**return** toplam;

}

**int** main() {

**int** x,y;

printf("Lutfen 2 adet sayi giriniz: \n");

scanf("%d %d",&x,&y);

printf("Sonucunuz:%d\n",toplam(x, y));

**return** 0;

}

gri olan bizim girdiklerimize ise argüman denir.

***Fonksiyon bildirimi:*** *(Fonksiyon tanımlamaktan farklı bir şeydir.)*

O da şu şekilde yapılır örnekle gösterelim.

int max(int num1, int num2);

#include <stdio.h>

**int** toplam(**int** n1,**int** n2); //***Fonksiyonu tanımlamadık ama bildirdik. Böylece main fonksiyon bittikten sonra tanımlamamız sıkıntı olmadı.***

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** x,y;

printf("Lutfen 2 adet sayi giriniz: \n");

scanf("%d %d",&x,&y);

printf("Sonucunuz:%d\n",toplam(x, y));

**return** 0;

}

**int** toplam(**int** n1,**int** n2){

**int** toplam=n1+n2;

**return** toplam;

}

***Void Fonksiyonlar***

*Void herhangi bir geri dönüş değeri olmayan fonksiyonlara denir. Void fonksiyon tanımladığında geri dönüş mekanizması kullanılmamalıdır. Bu return kullanılamayacağı anlamına gelmez. Çünkü returnun iki anlamı vardı:*

*1)Geri dönüş değeri döndermek.*

*2)Fonksiyonu orada sonlandırmak.*

örnek:

**void** fonk(**int** number){

**int** result= 2\*number+3;

printf("%d",result);

}

**int** main(**int** argc, **const** **char** \* argv[]) {

fonk(5);

**return** 0;

}

İlla argüman girmek zorunda da değiliz mesela 1 den 30 a kadar tek sayıları yazdıran bir kod yazalım:

#include <stdio.h>

**void** teksay(){

**int** i;

**for** (i=1; i<30; i+=2) {

printf("%d ",i);

}

}

**int** main(**int** argc, **const** **char** \* argv[]) {

teksay();

**return** 0;

}

**1 3 5 7 9 11 13 15 17 19 21 23 25 27 29**

***Inline fonksiyonlar***

*Kullanımı normal fonksiyonlarla aynıdır ancak daha az maliyet sağlar.*

*4 satırdan az kodlarda kullanılabilir ancak 4 satırdan fazla kodlarda kullanılması çok da tercih edilmez.*

***Variadic Fonksiyonlar***

*Değişken sayıda argüman ile çağrılabilen fonksiyonlardır.*

*Özellikle standart giriş çıkış işlemlerinde kullanılan fonksiyonlara denir.*

*int f (int x, int y, ...); //bu arada üç tane nokta varsa buna elipsis atomu denir*

*f(12, 13);*

*f(50, 100, 150);*

*f(20,25,30,35);*

*gibi.*

*Not: 1 ya da 0 rrgüman olursa syntax hatası olur.*

***Call By Referance and Call By Value***

***Call By Value***

#include <stdio.h>

**void** foo(**int** a){

a=200;

}

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** x=60;

foo(x);

printf("%d",x);

**return** 0;

}

*burada x in kopyası üzerinde bir değişiklik yapıldığından ekrana 200 değil 60 yazılacaktır.*

***Call By Referance***

#include <stdio.h>

**void** foo(**int** \*a){

\*a=200;

}

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** x=60;

foo(&x);

printf("%d",x);

**return** 0;

}

*burada ise x in adresini yolladığımızdan direkt x üzerinde değişiklik yapılır.*

*Örnek:*

#include <stdio.h>

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

**int** temp;

temp=x;

x=y;

y=temp;

}

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** a=20,b=30;

printf("a=%d b=%d \n",a,b);

swap(a,b);

printf("a=%d b=%d \n",a,b);

**return** 0;

}

**a=20 b=30**

**a=20 b=30**

Burada call by value kullanıldığından nesneler üzerinde değişiklik yapılmamış oluyor.

a ile bnin kopyaları üzerinde takas işlemi gerçekleştiğinden ekrana yazdırdığımız a ile b nin aynı kaldığını görüyoruz.

BU KONULAR POINTER KISMINDA DAHA DETAYLI ANLATILACAKTIR.

#include <stdio.h>

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

**int** temp;

temp=\*x;

\*x=\*y;

\*y=temp;

}

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** a=20,b=30;

printf("a=%d b=%d \n",a,b);

swap(&a,&b);

printf("a=%d b=%d \n",a,b);

**return** 0;

}

**a=20 b=30**

**a=30 b=20**

Burada ise call by referance kullanıldığından nesneler üzerinnde değişiklik yapılmış oluyor ve ekranda takas edilmiş hali görebiliyoruz.

**STANDART KÜTÜPHANE KAVRAMI ve KÜTÜPHANE OLUŞTURMA**

Bir header dosyasını birden fazla header dosyası ekleyebilir. Böyle bir durumda header içindeki tanımların tekrarlanmaması için “include guard”lar kullanılır.

#ifndef HEADER\_FILE

#define HEADER\_FILE

//code area

#endif

**SCOPE KAVRAMI VE NESNELERİN ÖMÜR KATEGORİSİ**

Bir nesne:

Otomatik ömürlü olabilir.

Statik ömürlü olabilir.

Dinamik ömürlü olabilir.

Belli bir kod aralığında hayatı başlayıp belli bir kod aralığından sonra işi bitene otomatik ömürlü diyoruz.

Program başladığında hayatı başlayan program bitince biten nesneye statik ömürlü diyoruz.

Programda istediğimiz zaman başlasın istediğimiz zaman bitsin istediğimiz nesnelere ise dinamik ömürlü nesne diyoruz.

#include <stdio.h>

//global name space

**int** y; //statik ömürlü bir değişken

**void** f(){

//local name space

**int** x;//otomatik ömürlü bir değişken

}

**int** main(**int** argc, **const** **char** \* argv[]) {

//local name space

**int** z; //otomatik ömürlü bir değişken

**return** 0;

}

//statik ömürlü değişkene herhangi bir atama yapmazsak derleyici otomatik olarak 0 atar. otomatik ömürlü değişkene herhangi bir değer girmezsek ise derleyici çöp bir değer atayacaktır.

Peki local name space’te statik ömürlü bir değişken oluşturabilir miyiz?

Evet. Bunun için static keyword’unu kullanacağız.

#include <stdio.h>

**void** f1(){

**int** x=20;

x++;

printf("x=%d\n",x);

}

**void** f2(){

**static** **int** x=20;

x++;

printf("x=%d\n",x);

}

**int** main(**int** argc, **const** **char** \* argv[]) {

printf("Otomatik omurlu olan:\n");

f1();

f1();

f1();

f1();

// Otomatik omurlu olanda fonksiyon bittigi an hayatı orda bittigi icin oyle bir degisken artık yoktur. Fonksiyon tekrar cagrıldıgında tekrar o degisken olusturulur, fonksiyon bittiginde tekrar biter. Bu sebeple her seferinde karsimiza 21 olarak gelir.

printf("Static omurlu olan:\n");

f2();

f2();

f2();

f2();

//Static omurlu olanda ise fonksiyon bittiginde hayatı sonlanmadigindan eski degerini korur, tekrar bir degisken tanımlamaz ve ekleyerek devam eder. bu sebeple karsimiza 21 22 23 24 olarak gelmistir.

//Bu durum performans acısından onemlidir cunku surekli bir yeni degisken olusturmamizi onler.

**return** 0;

}

**Otomatik omurlu olan:**

**x=21**

**x=21**

**x=21**

**x=21**

**Static omurlu olan:**

**x=21**

**x=22**

**x=23**

**x=24**

* Otomatik ömürlü nesneler bellekte stack bölümünde
* Static ömürlü nesneler data segment bölümünde
* Dinamik ömürlü nesnelerse heap bölümünde saklanır.

***SCOPE KAVRAMI (Bir ismin bilinirlik alanı)***

*C programlama dilinde 4 farklı scope türü vardır:*

* *FILE Scope*
* *Block Scope*
* *Function Prototype Scope*
* *Function Scope*

Global isim alanında bir değişken tanımlarsam bu FILE Scope’a tabiidir. Ve programın sonuna kadar o değişken tanınacaktır.

Block Scopeta ise o değişken bulunduğu bloğun içinde var olduğu satırdan bloğun son satırına kadar tanınır haldedir.

Function Prototype Scope: Oluşturduğumuz fonksiyondaki parametlerin olduğu kısımdır .

**int** f(**int** a, **int** b){

**return** 2;

}

Mesela burada int a, int a diyemeyiz.

Hatta fonksiyonun içinde de bir int a belirleyemeyiz.

#include <stdio.h>

**int** f(){

**int** z=10;

**return** 1;

}

**void** f2(){

**int** z=20;

}

**int** z=30;

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** z=40;

//mesela buraya z++ diyip yazdırırsak 41 yazar.

// ama 40 olmasaydı orada ve yine z++ deseydik bu sefer 31 yazardı.

**return** 0;

}

Mesela bu değişkenlerin hepsi birbirinden alakasızdır ve derleyici de sorun çıkarmaz.

İsim aramaya gelince ise bu eylem içten dışa yapılır.

***PREPROCESSORS / ÖNİŞLEMCİLER***

Text ile ilgili işlemler yapar.

# bu karakterle başlayan komutlar önişlemcilerle ilgilidir.

Bu komutlar 13 tane olup şunlardır:

#include

#define

#undef

#ifdef

#ifndef

#if

#else

#elif

#endif

#error

#pragma

# (null direktif, hizalama ve süslemede kullanılır başka bir olayı yok)

#line

***include direktifi***

Çoğunlukla kütüphane eklerken kullanılır.

< > default directory

“ “ actual directory (kendi oluşturduğumuz kütüphanelerde bunu kullanmazsak hata alırız.)

***Macro nedir?***

Simple macro

Kullanımı şu şekildedir:

#define isim yerine geçecek ifade

örnek:

#define SIZE 50

#define OFF 0 (mesela OFF gördüğü yere 0 yazacak)

#define ON 1

#define ALT\_SATIRA\_GEC ‘\n’

#define ERROR ‘Hatalı giriş! Tekrar deneyiniz.’

genelde 1 tab boş bırakarak yazılır ve büyük harf kullanılır.

#include <stdio.h>

#define KARAMAZOV\_KARDESLER 1110

#define BUDALA 1045

#define YERALTINDAN\_NOTLAR 200

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** kitap1=1423;

**if** (kitap1>BUDALA) {

printf("kitap1 uzun");

}

**else**

printf("kitap1 kısa ");

**return** 0;

}

Mesela aşağıdaki örnekte 0 yazdıracaktır.

#include <stdio.h>

#define BOOL int

#define FALSE 0

#define TRUE 1

**int** main(**int** argc, **const** **char** \* argv[]) {

BOOL x=**FALSE**;

printf("%d",x);

**return** 0;

}

**0**

**Function Like Macro**

Özellikle verim kritik alanlarda kullanılır.

Tek satırlık bir kod olmalıdır.

örnek:

#include <stdio.h>

#define max2(a,b) ((a)>(b)?(a):(b))

//ternary operation kullandık (koşul operatörü)

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** x=10,y=20;

**int** max=max2(x, y);

printf("max value: %d",max);

**return** 0;

}

main’in içini şöyle de yapabilirdin:

**int** x=10,y=20;

printf("max value: %d",max2(x,y));

örnek:

#include <stdio.h>

#define square(a) (a)\*(a)

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** x;

printf("please enter a number: ");

scanf("%d",&x);

printf("the number's square is: %d ",square(x));

**return** 0;

}

***Conditional Compiling (Koşullu Derleme)***

Önişlemci programı bazı komutları yürüterek kaynak dosyadaki bazı kodları verir bazılarını vermez. Böylece sıra derleyiciye geldiğinde kaynak dosyadaki kodların belli kısmını görür belli kısmını görmez. Buna koşullu derleme denir.

Bunu yapmamızın sebepleri nelerdir:

1. İşletim Sistemi Farklılığı
2. Donanımsal Farklılıklar
3. Derleyici Farklılığı
4. Dil Farklılığı
5. Versiyon Farklılığı
6. Lokalizasyon

#include <stdio.h>

#define VERSION 5

**int** main(**int** argc, **const** **char** \* argv[]) {

#if VERSION>3

printf("TRUE");

#else

printf("FALSE");

#endif

**return** 0;

}

Ancak burada iki önemli nokta var:

1)int sayı kullanılmalı

2) sarı kısımda sizeof(int) falan yazamayız.

#include <stdio.h>

#define VERSION 12

**int** main() {

#if VERSION>0 && VERSION<5

printf("kucuk");

#elif VERSION>5 && VERSION<10

printf("orta");

#elif VERSION>10 && VERSION<15

printf("buyuk");

#else

printf("nothing");

#endif

**return** 0;

}

ifdef ve ifndef de anlattı 2 dkda ama çok önemli değil gibi. (Videonun sonunda)

*#error ve #line komutları*

Önişlemci #error komutu görürse programı sonlandırır ve bizim error komutuyla verdiğimiz yazıyı yazdırır.

*#pragma*

Çok önemli değil lazım olursa bakarsın.

Predefined Symbolic Constants (Öntanımlı Sembolik Sabitler)

(Sembolik Sabit 🡪Macro idi)

Normal macrolarda başına #define koyup yapıyoduk bunda o olmayacak.

Bizi ilgilendiren en önemli önişlemci direktifleri, predefined komutlar şunlardır:

\_\_FILE\_\_ (iki alt tire yan yana)

\_\_LINE\_\_

\_\_DATE\_\_

\_\_TIME\_\_

\_\_func\_\_

\_\_STDC\_\_

1 )\_\_FILE\_\_ kaynak dosyanın adını yazdırabiliriz. String literali olduğundan %s ile yazdırdık.

**int** main() {

printf("%s\n",\_\_FILE\_\_);

**return** 0;

}

**/Users/asude/Desktop/PROFESYONEL C PROGRAMLAMA/kod deneme/kod deneme/main.c**

**Program ended with exit code: 0**

2) \_\_LINE\_\_ ile o kodun hangi satırda olduğunu yazdırabiliriz. int bir değer yazdırılacağından %d kullanılır.

printf("%d\n",\_\_LINE\_\_);

3)\_\_DATE\_\_ ile derleme işleminin yapıldığı tarihi yazdırabiliriz.

printf("%s\n",\_\_DATE\_\_);

**Mar 3 2021**

4) )\_\_TIME\_\_ ile derleme işleminin yapıldığı saati yazdırabiliriz.

5) )\_\_func\_\_ ile kod hangi fonksiyonun içindeyse onu yazdırabiliriz.

printf("%s\n",**\_\_func\_\_**);

**main**

***SABİTLER(CONSTANTS)***

*Tam Sayı Sabitleri*

*Gerçel Sayı Sabitleri*

*Karakter Sabitleri*

*String Sabitleri*

*Tam sayı sabitlerinin default türü* ***int****‘dir.*

*Gerçel sayı sabitlerinin default türü de double’dır.*

*c de şu sayı sistemleri vardır:*

*decimal-octal-hexadecimal*

*Ama ikili sayı sistemi yoktur. //extension olarak verilebilir.*

*Hexadecimal sayıları 0x ile %d ile yazdırırsak decimal halini görürüz.*

*örnek:*

printf("%d\n",0x12);

**18**

şu şekilde %x ile yazdırırsak da yazdığımız sayının hexadecimal karşılığını verir.

örnekler:

printf("%x\n",12);

**c**

printf("%x\n",20);

**14**

NOT: octal için de %o kullanılır.

ÖZET:

printf("%d\n",0xA5);

printf("%x\n",165); //x i büyük yazarsan A yazar.

165

a5

***CHARACTER TYPES***

*Tek tırnak içinde yazdığımız sabitlere karakter sabitleri denir.*

**int** main() {

**char** x='A';

printf("%d\n",x);//ASCII'deki sayısal karşılığı için %d

printf("%c\n",x);//Görüntüsü için %c

**return** 0;

}

**65**

**A**

**int** main() {

**char** x=65;

printf("%c\n",x); //Eger tek tırnak icinde olmadan sayısal karsılıgını girersek de yine %c ile goruntusunu yazdırabiliriz

**return** 0;

}

***ASCII’deki tüm karakterleri yazdıran bir kod yazarsak:***

#include <stdio.h>

**int** main() {

**int** i;

**for** (i=0; i<127; i++) {

printf("%d %c\n",i,i);

}

**return** 0;

}

**0**

**1 \_**

**2 \_**

**3 \_**

**4 \_**

**5 \_**

**6 \_**

**7 \_**

**8 \_**

**9**

**10**

**11 \_**

**12**

**13**

**14 \_**

**15 \_**

**16 \_**

**17 \_**

**18 \_**

**19 \_**

**20 \_**

**21 \_**

**22 \_**

**23 \_**

**24 \_**

**25 \_**

**26 \_**

**27 \_**

**28 \_**

**29 \_**

**30 \_**

**31 \_**

**32**

**33 !**

**34 "**

**35 #**

**36 $**

**37 %**

**38 &**

**39 '**

**40 (**

**41 )**

**42 \***

**43 +**

**44 ,**

**45 -**

**46 .**

**47 /**

**48 0**

**49 1**

**50 2**

**51 3**

**52 4**

**53 5**

**54 6**

**55 7**

**56 8**

**57 9**

**58 :**

**59 ;**

**60 <**

**61 =**

**62 >**

**63 ?**

**64 @**

**65 A**

**66 B**

**67 C**

**68 D**

**69 E**

**70 F**

**71 G**

**72 H**

**73 I**

**74 J**

**75 K**

**76 L**

**77 M**

**78 N**

**79 O**

**80 P**

**81 Q**

**82 R**

**83 S**

**84 T**

**85 U**

**86 V**

**87 W**

**88 X**

**89 Y**

**90 Z**

**91 [**

**92 \**

**93 ]**

**94 ^**

**95 \_**

**96 `**

**97 a**

**98 b**

**99 c**

**100 d**

**101 e**

**102 f**

**103 g**

**104 h**

**105 i**

**106 j**

**107 k**

**108 l**

**109 m**

**110 n**

**111 o**

**112 p**

**113 q**

**114 r**

**115 s**

**116 t**

**117 u**

**118 v**

**119 w**

**120 x**

**121 y**

**122 z**

**123 {**

**124 |**

**125 }**

**126 ~**

***Harflerin ASCII’de nerede başladığını bilmek zorunda değilsin, forun içine şu şekilde de koyabilirsin. Burada char yerine int demek ve yine %d %c ile yazdırmak da aynı sonucu veriyor aklında bulunsun.***

**int** main() {

**for** (**char** x='A'; x<='Z'; x++) {

printf("%d: %c\n",x,x);

}

**return** 0;

}

**65: A**

**66: B**

**67: C**

**68: D**

**69: E**

**70: F**

**71: G**

**72: H**

**73: I**

**74: J**

**75: K**

**76: L**

**77: M**

**78: N**

**79: O**

**80: P**

**81: Q**

**82: R**

**83: S**

**84: T**

**85: U**

**86: V**

**87: W**

**88: X**

**89: Y**

**90: Z**

***Escape Sequence***

*\ karakteriyle bazı karakterler yan yana geldiğinde özel anlam ifade eden bazı sabitler oluşur*

\n

\t

\a

\r

\b

\0

\\

çeşitli anlamları var yazmaya üşendim ☹ çok da önemli değil.

***Ctype ve limit kütüphanesi***

**Karakter türleri:**

* Printable characters: görüntüsü olan karakterler A b ! 98 \* gibi
* Control characters: görüntüsü olmayan karakterler: mesela space tuşu ( ), ya da \n \t gibi karakterler.
* alphanumeric characters: harf veya rakam karakterleri a b c 1 2 3
* Punctuation characters: görüntüsü olup alphanumeric olmayan karakterler + - \* ; / gibi

ctype kütüphanesi:

bu kütüphaneyi 2 amaç için kullanacağız:

1. karakter dönüşümü
2. karakter test fonksiyonu

örnek:

#include <stdio.h>

#include <ctype.h>

**int** main() {

**int** c;

printf("Lutfen bir karakter giriniz: \n");

c=getchar();

**if** (isupper(c)) {

printf("Evet %c bir buyuk harf karakteridir.\n",c);

}

**else**

printf("Hayir, %c bir buyuk harf karakteri degildir.\n",c);

**return** 0;

}

**Lutfen bir karakter giriniz:**

A

**Evet A bir buyuk harf karakteridir.**

*(Kütüphaneye dair “All the functions accepts****int****as a parameter” şeklinde bir bilgi olduğundan int kullandık ancak char kullanarak denediğimde de hata almadım.)*

*Tüm fonksyionları gösteren bir örnek:*

#include <stdio.h>

#include <ctype.h>

**int** main() {

**char** c;

printf("Please enter a character: \n");

c=getchar();

**if** (isupper(c)) {

printf("Yes %c is a upper character.\n",c);

}

**if** (isalnum(c)) {

printf("Yes %c is a alphanumeric character.\n",c);

}

**if** (isxdigit(c)) {

printf("Yes %c is a hexadecimal character.\n",c);

}

**if** (isdigit(c)) {

printf("Yes %c is a digit character.\n",c);

}

**if** (isprint(c)) {

printf("Yes %c is a printable character.\n",c);

}

**if** (isgraph(c)) {

printf("Yes %c is a graph character.\n",c);

}

**if** (ispunct(c)) {

printf("Yes %c is a punct. character.\n",c);

}

**if** (iscntrl(c)) {

printf("Yes %c is a control character.\n",c);

}

**if** (isalpha(c)) {

printf("Yes %c is a alphabetic character.\n",c);

}

**return** 0;

}

örnekler:

**Please enter a character:**

A

**Yes A is a upper character.**

**Yes A is a alphanumeric character.**

**Yes A is a hexadecimal character.**

**Yes A is a printable character.**

**Yes A is a graph character.**

**Yes A is a alphabetic character.**

**Please enter a character:**

9

**Yes 9 is a alphanumeric character.**

**Yes 9 is a hexadecimal character.**

**Yes 9 is a digit character.**

**Yes 9 is a printable character.**

**Yes 9 is a graph character.**

**Please enter a character:**

.

**Yes . is a printable character.**

**Yes . is a graph character.**

**Yes . is a punct. character.**

***Diğer iki fonksiyona gelirsek:***

*int toupper(int c) //küçük girdiğimiz harfi büyüğe dönüştürecek*

*int tolower(int c) // büyük girdiğimiz harfi küçüğe dönüştürecek*

**int** main() {

**char** c;

printf("Please enter a character: \n");

c=getchar();

printf("%c %c\n",c, toupper(c));

**return** 0;

}

**Please enter a character:**

a

**a A**

**int** main() {

**char** c;

printf("Please enter a character: \n");

c=getchar();

printf("%c %c\n",c, tolower(c));

**return** 0;

}

**Please enter a character:**

D

**D d**

**DİZİLER**

Dizideki tüm elemanlara şu şekilde ilk atamayı 0 olarak yapabilirsin.

#include <stdio.h>

#define SIZE 10

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** i;

**int** a[SIZE]={0};

**for** (i=0; i<SIZE; i++) {

printf("%d ",a[i]);

}

**return** 0;

}

**int** a[SIZE]={1};

yaparsan da ilk elemana 1 koyar gerisini yine 0 yapar yani komple 1 yapmıyor.

srand(time(NULL)) ile dizide rastgele değerler elde etme.

#include <stdio.h>

#define SIZE 10

**int** main(**int** argc, **const** **char** \* argv[]) {

srand(time(**NULL**));

**int** a[SIZE];

**int** i;

**for** (i=0; i<SIZE; i++) {

a[i]=rand()%50;

}

**for** (i=0; i<SIZE; i++) {

printf("%d ",a[i]);

}

**return** 0;

}

***Pointer notlar***

//

// main.c

// pointer giriş

//

// Created by Asude Ekiz on 28.10.2021.

//

#include <stdio.h>

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** \*ip, x; //int bir pointer ve degisken tanimladik

x=21; //x e deger atadik

ip=&x; // pointera da x degiskeninin adresini atadık

printf("%d\n", x); //x i direkt yazdirdik

printf("%d\n", \*ip); // \*ip yaparak pointerın tuttugu adresteki degeri, yani x i yazdirdik.

//calistirinca her ikisi de 21 olmali

//adreslere gecelim

printf("%p\n", &x);// &kullanarak x in adresini yazdiralim

printf("%p\n", ip); //pointerı kullanarak x in adresini yazdiralim

//isaretciler yoluyla degiskenlere deger atayabiliriz

**int** y=10; //baslangıcta boyle olsun

**int** \*ip2=&y;

printf("%d\n",y);//10

printf("%d\n",\*ip2);//10

printf("%p\n", &y);//y in adresi

printf("%p\n",ip2);// y in adresi

//y in adresindeki degeri pointer vasitasiyla degistirelim

\*ip2=15 ;//olsun (yani dedik ki y yi gosteren pointerin ici artık 10 degil 15 olsun.)

//simdi y i yazdiralim

printf("%d\n", y);

//ama y nin adresini degistirmedik

printf("%p\n",&y);

printf("%p\n",ip2);

printf("%d\n",\*&y);//y nin adresine git adresteki degeri yaz

printf("%d\n",\*ip2) ;//y nin pointerına git ordaki degeri yaz

**return** 0;

}

2.ders

#include <stdio.h>

**int** main(**int** argc, **const** **char** \* argv[]) {

//ONEMLİ NOT

// int \*ptr derkenki yildiz ile sonradan \*ptr yapman ayni sey degil

// int \*ptr yaparkenki yildiz sadece bi pointer olarak tanimladigimizi belirtiyo

/\*

//simdi once f tanimlayalim, dolayli olarak degerini belirleyelim sonra point edelim

int f;

\*&f=4; //f in degerini f in adresinin degeri,,, diyerek tanimladik dolayli yoldan yani

int \*ip4=&f;

printf("%d\n",f);//4

printf("%d\n",\*ip4);//4

printf("%p\n", &f);//f in adresi

printf("%p\n",ip4);// f in adresi

\*/

//burda da m tanimlayalim ama deger bildirmeden point edelim ve icerigini soyle diyelim?? nolcak

**int** m;

**int** n;

**int**\* ip4=&m;

**int**\* ip5=&n;

printf("%d\n",\*ip4);//0 yazdırır cunku icerigi bilmiyo ki dsjgjk

printf("%d\n",\*ip5);//yine 0 mesela

\*&m=5; //simdi dolayli yoldan m in degerini bildirdik

printf("%d\n",\*ip4);//burda 5 yazdirir artik cunku biliyo

printf("%d\n",m);//5

\*&n=6;// yine dolayli yoldan n i bildirdik

printf("%d\n",\*ip5);//burda 6 yazdirir artik cunku biliyo

printf("%d\n",n);//6

printf("%p\n", &m);//m in adresi

printf("%p\n",ip4);// m in adresi

**return** 0;

}

3.ders

#include <stdio.h>

**int** main(**int** argc, **const** **char** \* argv[]) {

//adresi direkt kendimiz belirleyelim

//16F8A olsun

**char** \*ptr= (**char** \*)0x16f8a;

**char** \*ptr2= (**char** \*)0x7ffeefbff43c;

printf("ptr: %p\n",ptr);

printf("ptr2: %p\n",ptr2);

//o kafamiza gore verdigimiz adrese atama yapalim ve o degeri yadirmaya calisalim bakalim erisimimizin

//oldugu bir adres miymis

\*ptr=20;

printf("\*ptr: %d\n",\*ptr); // hata veriyor demek ki ulasabilecegimiz bi adres degilmis

//daha once baska bi kodda x diyip onun adresini yazdirmistim o adresi atadigimiz ptr2 ye deger atayalim

//ulasilabilir bi adres oldugundan o degeri gosterecek

\*ptr2=30;

printf("\*ptr2: %d\n",\*ptr2);

**return** 0;

}

Pointerlar neden kullanılır?

* call by reference
* dizileri fonksiyona aktarmanın tek yolu
* bir fonksiyon birden fazla değer döndüremez. ancak pointerlar sayesinde birden fazla değer aktarmak mumkun
* storafe’ı buyuk olan türlere pointer parametre olmayan fonksiyonlara argüman geçmek maliyetli ve yavaş bir işlemdir. pointer parametre kullanılarak maliyet ve performans artışı sağlanabilir
* bellek bölgesine doğrudan erişim
* pointerların doğru kullanımıyla performans ve maliyet artışı sağlanabilir

#include <stdio.h>

**void** func(**int** a,**int** b){ //call by valueye ornek bi fonksiyon

**int** temp;

temp=a;

a=b;

b=temp;

}

**void** func2(**int** \*a, **int** \*b){ //cal by reference'e ornek bi fonksiyon

**int** temp;

temp=\*a;

\*a=\*b;

\*b=temp;

//fonksiyonun icine ADRES yollicaz

//sonra temp zaten int deger, pointer degil, ona \*a yaparak bir deger yollicaz cunku burda int \*a diyerek

// bir pointer tanimladik (sey gibi iste int \*ptr bi pointer, sonra \*ptr yaparsan degere ulasirsin)

//a adresinin degerini b adresinin degerine atadik sonra b adresinin degerine de tempi atadik swap gerceklesti

}

**int** main(**int** argc, **const** **char** \* argv[]) {

//call by value

**int** a=20;

**int** b=30;

printf("a= %d b=%d\n",a,b);

func(a, b);

printf("a= %d b=%d\n",a,b);

// a ile b nin degeri degismedi CUNKU o void fonksiyonunda kopyalari uzerinde swap islemi yapıldı.

//call by reference

func2(&a, &b);

//bu fonksiyona ADRESLERİ yolladik ve adreslerin degerini swapleyerek

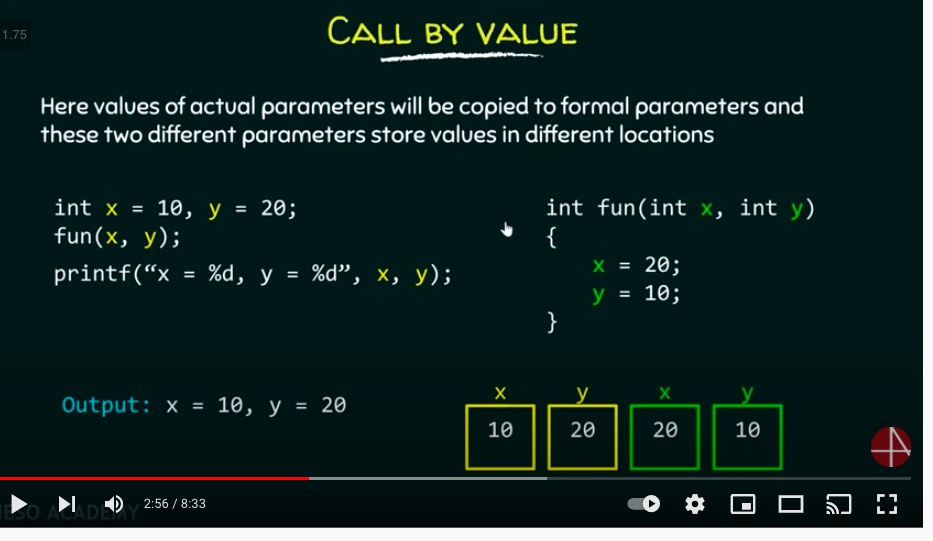
//kopyasi degil gercek degerler uzerinde oynamis olduk

printf("a= %d b=%d\n",a,b);

//burda artik swaplenmis halini yazicak

**return** 0;

}



metin, ekran, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

#include <stdio.h>

**int** fonk(**int** a,**int** b,**int** c){

**int** toplam;

toplam=a+b;

c=2\*c;

**return** toplam;

}

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** x=20;

**int** y=30;

**int** z=10;

printf("x: %d y: %d z:%d\n",x,y,z);

printf("toplam: %d\n", fonk(x, y, z));

printf("x: %d y: %d z:%d\n",x,y,z);

//FONKSİYON VE CALL BY VALUE'YA ORNEK

**return** 0;

}

NULL POINTER

Hiçbir nesneyi işaret etmeyen pointer

#define NULL ((char \* )0)

#define NULL 0

#define NULL 0L

üçü de aynı şey

#include <stdio.h>

**int** \*ptr2;

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** \*ptr=**NULL**;

printf("%p\n", ptr); //0 adresini gosterecek yani hicbir nesneyi isaret etmiyor.

printf("%p\n", ptr2); //bu da 0 adresini gosterecek cunku global isim alanında olusturdugumuz pointerlar da aslında null pointerdır.

**return** 0;

}

// printf("%d\n",\*ptr);

// bir yeri göstermeyen pointerin icindeki degeri ogrenmek istersek hata aliriz

// printf("%d\n",\*NULL); //bunda da ayni sekilde

//\*NULL=20;

//erisemedigimiz yere atama yapsak da yine hata

// printf("%d\n",\*NULL);

***Bazı pointer durumlari***

//bildiğim seylerdi

***Pointer döndüren fonksiyonlar***

Geri dönüş değeri pointer olan fonksiyonlar

#include <stdio.h>

**int**\* square(**int**\* ptr){ //pointer dönderen bir fonksiyon tanımladık

\*ptr=(\*ptr)\*(\*ptr); //tanım yaptıktan sonra \*ptr yaptığımızda değerden bahsediyoduk bu satırda kare aldık

**return** ptr; //ve sonucun adresini dönderdik

}

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** x=5;

**int**\* p=square(&x);//eşitliğin sağında call by reference ile x in adresini yolladik ve dönen adresi başka bir

//pointerda tuttuk.

printf("%d\n",\*p); //sonra da son pointerın değerini %d ile yazdırarak 25 sonucuna ulaştık.

**return** 0;

}

#include <stdio.h>

**int**\* sum(**int**\* s1,**int**\* s2){

**static** **int** result; //adresle donen fonksiyonlarda otomatik omurlu nesnelerle dönmememiz lazım

result=\*s1 + \*s2;

**return** &result;

}

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** a=10;

**int** b=20;

**int**\* ptr= sum(&a, &b);

printf("%d\n", \*ptr);

**return** 0;

}

***Array to pointer converter***

int a[5] diye bir dizimiz olsun.

Herhangi bir yerde a yazmamız &a[0] yazmamızla aynı şeydir ve dizinin ilk elemanının adresini belirtir.

bu sebeple int\* ptr = a gibi bir atama yapmamız mümkündür.

#include <stdio.h>

**int** main(**int** argc, **const** **char** \* argv[]) {

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

**int**\* p1=a;

**int**\* p2=&a[0];

printf("p1: %p\n",p1);//a[0] ın adresini göstermeli

printf("p2: %p\n",p2);//bu da a[0] ın adresini göstermeli

printf("\*p1: %d\n",\*p1);//a[0]ı yani 1 i göstermeli

printf("\*p2: %d\n",\*p2);//a[0]ı yani 1 i göstermeli

**return** 0;

}

**p1: 0x7ffeefbff4d0**

**p2: 0x7ffeefbff4d0**

**\*p1: 1**

**\*p2: 1**

***POINTER ARİTMETİĞİ***

* Bir pointer ile bir sayı toplanabilir
* Bir tam sayı ile bir pointerı toplanabilir
* Bir pointerdan bir tam sayıyı çıkarılabilir

(Bir tam sayıdan bir pointer çıkarılmaz)

* Bir pointerdan başka bir pointerı çıkarılabilir

#include <stdio.h>

**int** main() {

**int** a[]={2,4,6,8,10};

**int**\* ptr=a;

ptr=a+2; //dizinin 0.elemanının adresini tutuyodu şimdi 2 ileriki aelemanın adresini tutuyor.

printf("%d\n",\*ptr); // \*ptr yapınca da a[2] yi gösterdi

ptr-=1;//1 çıkaralım, bu sefer 4 ün adresini işaret ediyor.

printf("%d\n",\*ptr);

printf("ptr: %p\n",ptr); //4 ün adresini öğrenelim

ptr=ptr-1;//ilk elemanın adresine geri götürelim

printf("ptr: %p\n",ptr);

//şimdi 1 adım daha geriye gitsin bakalım hangi adresi ve hangi değeri gösterecek

ptr=ptr-1;

printf("ptr (ilk değerden bir adım geri): %p\n",ptr);

printf("\*ptr(ilk değerden bir adım gerideki değer): %d\n",\*ptr); //bnde 0 çıktı ama çöp değer de çıkabilirdi

//

**return** 0;

}

**6**

**4**

**ptr: 0x7ffeefbff4d4**

**ptr: 0x7ffeefbff4d0**

**ptr (ilk değerden bir adım geri): 0x7ffeefbff4cc**

**\*ptr(ilk değerden bir adım gerideki değer): 0**

**Program ended with exit code: 0**

#include <stdio.h>

**int** main() {

**int** a[]={2,4,6,8,10};

**int**\* p1=a;

**int**\* p2=a+2;

**int**\*p3=p1+p2; //GECERSİZ İSLEM

printf("%p\n",p3);//calismadi bile

printf("%d\n",\*p3);//bu da calismadi

**return** 0;

}

bunu sor

#include <stdio.h>

**int** main() {

**int** a[]={2,4,6,8,10};

**int**\* p1=a;

**int**\* p2=a+8;//a’nın yanındaki eleman kaçsa onu döndürüyor??(0x8, 0x4 gibi)

**int** \*p3=p2-p1;

printf("%p\n",p3);

printf("%d\n",\*p3);

**return** 0;

}

***DİNAMİK BELLEK YÖNETİMİ***

***önce dinamik bellek yönetiminde kullanılan void pointer neydi ona bakalım sonra dinamik bellet yönetimi fonksiyonlarına geçelim***

#include <stdio.h>

**int** main(**int** argc, **const** **char** \* argv[]) {

//void pointer nedir??

//türü belli olmayan pointer

//başta tanımlarken belli değil ama kullanırken tipini belli etmemiz gerekiyor

/\* Giriş örneği

int a=10;

void \*ptr=&a;

printf("%d\n",\*((int\*)ptr));//pointerın int değer tuttuğunu söylememiz gerekiyor

\*/

// şimdi int char ve double değşikenler belirleyelim.

//tek bir pointerla (sonrasında yazdırırken türünü belli ederek yerlerini tutucaz?)

**int** x=20;

**char** y='a';

**double** z=2.5;

**void** \*veri;

veri=&x;

printf("%d\n",\*((**int** \*)veri));

printf("%p\n",veri);

veri=&y;

printf("%c\n",\*((**char** \*)veri));

printf("%p\n",veri);

veri=&z;

printf("%.2lf\n",\*((**double** \*)veri));

printf("%p\n",veri);

//tek pointer kullandık ama tabii yeni bi türe geçtikçe yeri değişti

**return** 0;

}

**20**

**0x7ffeefbff3fc**

**a**

**0x7ffeefbff3fb**

**2.50**

**0x7ffeefbff3f0**

**Program ended with exit code: 0**

Malloc:

Başarılı olması durumunda heap alanından ne kadarlık bir bellek alanı oluşturulacak. Başarısız olursa oluşturmaz ve geri dönüş değeri Null pointer olur.

Calloc:

Malloc’un özel bi biçimi.

Bellek bloklarını siliyor yani bitleri sıfırlarla dolduruyor.

Realloc:

Yeniden allocation. Daha önceden elde edilmiş bellek bloğunun bellek bloğunu büyütmeye ya da küçültmeye yarar.

Free:

Serbest bırakmak. Kullanılan dinamik ömürlü bellek alanının geri verilmesine denir.

malloc:

#include <stdio.h>

#include <stdlib.h>

**int** main(**int** argc, **const** **char** \* argv[]) {

//void\* malloc(size\_t \_size)

**int** \*ptr= (**int**\*)malloc(20);//malloc fonksiyonu void \* seklindeydi onu int \* a dönüştürdük

**if** (ptr==**NULL**) {

printf("bellek yetersiz\n");

exit(EXIT\_FAILURE);

}

printf("bellek yeterli\n");

printf("\n ayrılan bellegin ilk degeri garbage iken: %d\n",\*ptr);//benim derleyicim 0 koyuyor

\*ptr=10;

\*(ptr+1)=20;//pointer aritmetiği

ptr[2]=30;//\*(ptr+2) yerine böyle de yazabilirim

printf("\n\*ptr= %d\n \*(ptr+1)=%d\n \*(ptr+2)=%d\n",\*ptr,\*(ptr+1),\*(ptr+2));

**return** 0;

}

calloc:

ayrılan alanda mallocta olduğu gibi garbage değer değil 0 olacak.

bir diğer farkı da malloc tek parametre alıyordu. malloc(40) gibi mesela (ama malloc 10\*sizeof(int)) şeklinde de yazabilirsin.

callocta ise iki parametre alması zorunlu

void \*calloc(N\*sizeof()) şeklinde

ör:

void \*calloc(8\*sizeof(double)) gibi

#include <stdio.h>

#include <stdlib.h>

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** N;

printf("How many elements in this array?\n");

scanf("%d",&N);

**int** \*iptr= (**int** \*)calloc(N, **sizeof**(**int**));

**if** (!iptr) {

printf("Bellek yetersiz\n");

**return** 1;

}

printf("\n");

**for** (**int** i=0; i<N; i++) {

printf("%d ",iptr[i]);

}

free(iptr);

printf("\n");

**return** 0;

}

örnek:

//

// main.c

// pointer malloc karisik

//

// Created by Asude Ekiz on 9.11.2021.

//

#include <stdio.h>

#include <stdlib.h>

**int** \*makeArray(size\_t size, **int** val);

**int** main(**int** argc, **const** **char** \* argv[]) {

**int** \*p= makeArray(5, 30);// burdaki 5 dizi boyutu, yani 5 byte değil. 5 tane içinde bir int olan kutu gibi düşün yani bir kutuda 4 byte var. ve onları 30 ile doldursun istedik

//dışarıdan da alabilirdik

**for** (**int** i=0; i<5; i++) {

printf("%d ",p[i]);

}

printf("\n");

//burda da size ile value'yi disardan alalım;//bu kısmı ben ekledim kurstakine ek

**int** size;

**int** val;

printf("Lutfen dizinin boyutunu giriniz: \n");

scanf("%d",&size);

printf("Dizi hangi degerle doldurulacak?\n");

scanf("%d",&val);

**int**\*p2=makeArray(size, val);

**for** (**int** i=0; i<size; i++) {

printf("%d ",p2[i]);

}

**return** 0;

}

**int** \*makeArray(size\_t size, **int** val){

**int** \*ptr=(**int**\*) malloc(size\***sizeof**(**int**));

**if** (ptr!=**NULL**) {

**for** (**int** i=0; i<size; i++) {

ptr[i]=val;

}

}

**return** ptr;

}

#include <stdio.h>

#include <stdlib.h>

**int** main() {

**int** \*a=(**int**\*)malloc(20\***sizeof**(**int**));

**int** \*b=(**int**\*)calloc(20, **sizeof**(**int**));

**int** i;

**for** (i=0; i<20; i++) {

printf("b[%2d]: %d a[%2d]:%d \n",i,b[i],i,a[i]);

}

free(a);

free(b);

**return** 0;

}

**b[ 0]: 0 a[ 0]:0**

**b[ 1]: 0 a[ 1]:0**

**b[ 2]: 0 a[ 2]:0**

**b[ 3]: 0 a[ 3]:0**

**b[ 4]: 0 a[ 4]:16**

**b[ 5]: 0 a[ 5]:0**

**b[ 6]: 0 a[ 6]:0**

**b[ 7]: 0 a[ 7]:1**

**b[ 8]: 0 a[ 8]:0**

**b[ 9]: 0 a[ 9]:0**

**b[10]: 0 a[10]:271077928**

**b[11]: 0 a[11]:-1610612736**

**b[12]: 0 a[12]:102**

**b[13]: 0 a[13]:0**

**b[14]: 0 a[14]:0**

**b[15]: 0 a[15]:131072**

**b[16]: 0 a[16]:1886938408**

**b[17]: 0 a[17]:1835627109**

**b[18]: 0 a[18]:1635020389**

**b[19]: 0 a[19]:1634557804**

**Program ended with exit code: 0**