**VERİ TİPLERİ**

**İnteger %d**

**Float %f**

**Double %lf**

**Char %c**

**%.2lf virgülden sonra sadece iki rakam yazdırılmak istendiğinde**

**%3.lf virgülden önce sadece üç rakam yazılmak istendiğinde**

**// yorum**

**/\* \*/ yorum**

**gcc -std=c99 -Wall -Werror week01.c -o week01**

**const int sabit=1; değiştirilemez sabittir.**

**İnt değişken=0; değiştirile bilir.**

**Scanf(“%d”,&değişken); scanfte herzaman virgülden sonra & yazılır çünkü scanf depolama alannına göndermede bulunur**

**OPERATÖRLER**

**- , + , \* , /**

**/ bölme işleminde her iki sayıda double olursa sonuç tam çıkar.**

**Double ondalıklı\_sayi\_bolme\_işlemi = sayı1 / (double) sayı2;**

**ARTTIRMA-AZALTMA İÇİN ++ / - - KULLANILIR.**

**ATAMA OPERATÖRLERİ**

**= TEMEL ATAMA OPERATÖRÜ**

**a += b demek a = a+ b demektir**

**a -= b demek a= a-b demektir**

**a /= b demek a= a/b demektri;**

**a \*= b demek a= a\*b demektir**

**a %=b demek a= a%b demektir.**

**TEMEL KÜTÜPHANELER & STANDART FONKSİYONLAR**

**<stdio.h> Temel Input(scanf) / Output(printf) fonksiyonlarını içerir**

**<stdlib.h> EXIT\_SUCCESS VE EXIT\_FAILURE return kalıplarını içerir**

**<math.h> Yaygın olarak kullanılan matematiksel fonksiyonları içerir.(karekök üstel falan)**

**ONEMLİ NOT= <math.h> kullanıldığı zaman derleme komutunun sonuna -lm yazılmalıdır.**

**Gcc -std=c99 -Wall -Werror week01.c -o week01 -lm**

**IF /ELSE / ELSE İF**

**İf(cdsc){printf(“%d”,sen);}**

**Else İf(cdsc){printf(“%d”,sen);}**

**Else{printf(“%d”,sen);}**

**== , !=, >= , <= , < , > operatörleri kullanılır**

**BOOLEAN TRUE FALSE**

**Boolean kullanmak için <stdbool.h> kütüphanesi kullanılır.**

**true false den başka bir şey değildir**

**if conditionun içindeki değerler true false yani boolean dönüştürülür.**

**True=1 false=0 dır**

**Printf(“%d”, false); bu sıfıra eşittir**

**Printf(“%d”, trur); bu bire eşittir**

**Tek bir if else bloğunda fazla koşul kullanmak için || && ! gibi ifadeler kullanılır**

**|| veya mantıksal operatörüdür**

**&& ve mantıksal operatörüdür**

**! değil mantıksal operatörüdür**

**FOR WHİLE(DÖNGÜLER)**

**Break bir döngünün istenilen koşula göre sonlanmasını sağlar**

**Continue bir döngünün istenilen bir durumda es geçilmesini sağlar**

**RASTGELE SAYI ÜRETME VE <TİME.H> KÜTÜPHANES**

**Rastgele sayılara zar atma, loto vb şans oyunlarında ihtiyaç duyulur. Rastgele sayılar üretmek için kullanılması gerekenler.**

**<stdlib.h> kütüphanesindeki rand() fonksiyonu**

**<stdlib.h> kütüphanisndeki srand() fonksiyonu**

**<time.h> kütüphanesindeki time() fonksiyonu**

**Rand()**

printf("%d ",rand() % ust\_sinir); // for rand()

**srand() time()**

srand(time(NULL));

printf("%d ",rand() % ust\_sinir);

Alt ve üst sınırı belirli olan bir sayı üretmek için yazı düzeni şu şekildedir:

**rand() % (üst sınır - alt sınır + 1)) + alt sınır**

**FONKSİYONLAR**

**STANDART FUNCTİONS**

<stdlib.h> kütüphanesinde abs(int value) fonksiyonu tam sayının mutlak değerini bulmak için kullanılır.

**KULLANICI TANIMLI FONKSİYONLAR**

NOT= C kodları derleyici tarafından yukarıdan aşağıya doğru okunur. C dilinin bir kuralı olaraktan fonksiyonlar mainden önce yazılmalıdır. Eğer üstüne değil altına yazılırsa bu fonksiyonların prototipleri yazılmalıdır üste.

**LOKAL DEĞİŞKENLER** :Fonksiyoun içinde tanımlanmış ve sadece tanımlandığı fonksiyonlar içirişinde kullanılabilen yerel değişkenlerdir.

**GLOBAL DEĞİŞKENLER**: Fonksiyonların dışında tanımlanmış ve tüm fonksiyonlar tarafından kullanılabilen evrensel değişkenlerdir.

**PARAMETRE OLARAK TAIMLANMIŞ DEĞİŞKENLER:** Fonksiyon parametresi olarak tanımlanan, fonksiyon çağrılırken kendisine bir değer atanan ve sadece tanımlandığı fonksiyon içerisinde kullanılabilen değişkenlerdir

**MACRO SABİTLER:** #define anahtar kelimesiyle kullanılan, büyük harflerle adlandırılan ve değiştirilemez bir değere sahip sabitlerdir(const) . Makrolar globaldir ve tüm fonksiyonlar tarafından kullanılabilir.

**DİZİLER ARRAY**

Diziler belirli bir veri tipine ait birden fazla değeri depolayabilen yapılardır. Her bir dizinin bünyesinde depoladığı değişmez bir veri tipi vardır. Yani bir dizi birden fazla veri tipini depolayamaz. Diziler en fazla belirlenen sayıda veriyi depolayabilirler. Bu kapasite değeri ya dizi tanımlanırken belirlenir ya da diziye belirli bir küme atanarak otomatik olarak belirlenir.

veriTipi diziAdi[diziKapasitesi];

veriTipi diziAdi[] = {veriKumesi};

. Diziler verileri belleğin bir kısmında ardışık olarak depolar.

**Dizilerin Fonksiyonlarla Kullanımı**

Diziler aynı zamanda veri tiplerinde olduğu gibi fonksiyonlarla beraber kullanılabilir. Yani fonksiyonun parametresi bir dizi olabilir. Bu işlem yapılırken dizinin büyüklüğü de fonksiyona parametre olarak gönderilmelidir.

**Scanning an array**

int size = 10;

int arr[size];

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

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

}

## 1 FİNDİNG LENGTH OF AN ARRAY uzunluk

for(int i=0; array\_1[i]!='\0'; i++)

{

length++;

}

## 2 USAGE OF ARRAY İN FUNCTİONS

printf("%lf",ortalama(array\_1,length));

double ortalama(double \* arr, int size)

## 3 ARRAY ORTALAMA

for(int i=0;arr[i]!='\0';i++)

{

toplam=toplam+arr[i];

number++;

}

**2 BOYUTLU DİZİLER ARRAY**

## 1 PRİNTİNG 2 BOYUTLU DİZİLER

**double array\_1[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};**

**for(int i=0;i<3;i++)**

**{**

**for(int j=0;j<4;j++)**

**{**

**printf("%.1lf ",array\_1[i][j]);**

**}**

**printf("\n");**

**}**

## 2 FONKSİYONLAR VE 2 BOYUTLU DİZİLER(2 DİMENSİONAL ARRAYS İN FUNCTİONS)

r. Burada en çok dikkat edilmesi gereken nokta iki boyutlu dizinin sütun büyüklüğüdür. Sütun büyüklüğü sabit bir sayı olarak belirlenmelidir ve dizinin belirtildiği her yerde sütun değeri de kendisine ait köşeli parantez içerisinde belirtilmelidir. Bunu yaparken sütun değerini macro olarak belirlemek tavsiye edilen bir yöntemdir. Sütun değeri macro olarak belirlendikten sonra artık değiştirilemez bir değerdir ve tüm fonksiyonlar tarafından kullanılabilir. Bu durumda iki boyutlu diziyi fonksiyona gönderirken beraberinde sadece satır değeri gönderilir.

void deger\_yazdir(int satir,double array\_1[][SUTUN])

int satir=5;

deger\_yazdir(satir, dizi);

void findintwodimaaray(int satir, double array\_1[][SUTUN], double finded)

int searchednumber=4;

findintwodimaaray(satir, dizi, searchednumber);

## 3 FİNDİNG İ TWO DİMENSİOANL ARRAY

**void findintwodimaaray(int satir, double array\_1[][SUTUN], double finded)**

**{**

**for(int i=0; i< satir; i++)**

**{**

**for(int j=0; j< SUTUN;j++)**

**{**

**if(array\_1[i][j]==finded)**

**{**

**printf("The number has been finded in %d, %d ",i,j);**

**return;**

**}**

**}**

**}**

**printf("The number has not been finded");**

**}**

**DİZİLERDE ARRAYLERDE ALGORİTHMALAR**

## 1 EN KÜÇÜK DEĞERİ BULMA

**int min\_deger(int array[], int size)**

**{**

**int minnumber=array[0];**

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

**if(minnumber>array[i]){**

**minnumber=array[i];**

**}**

**}**

**return minnumber;**

**}**

## 2 EN BÜYÜK DEĞERİ BULMA

**int max\_deger(int array[], int size)**

**{**

**int maxnumber=array[0];**

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

**if(maxnumber<array[i]){**

**maxnumber=array[i];**

**}**

**}**

**return maxnumber;**

**}**

## 3 KÜÇÜKTEN BÜYÜĞE SIRALMA

**void kucukten\_buyuge(int array[], int size){**

**for(int i=0; i<size-1;i++){**

**for(int j=0; j<size-i-1;j++){**

**if(array[j]>array[j+1]){**

**int changingvalue= array[j];**

**array[j]=array[j+1];**

**array[j+1]=changingvalue;**

**}**

**}**

**}**

**}**

## 4 BÜYÜKTEN KÜÇÜĞE SIRALMA

**void buyukten\_kucuge(int array[], int size){**

**for(int i=0; i<size-1;i++){**

**for(int j=0; j<size-i-1; j++){**

**if(array[j]<array[j+1]){**

**int changingvalue=array[j+1];**

**array[j+1]=array[j];**

**array[j]=changingvalue;**

**}**

**}**

**}**

**}**

**POİNTERS**

**Not:** Bir adres değeri yazdırılmak istendiğinde format olarak **%p** formatı kullanılır.

**## 1 POİNTER YAZMAK**

**İnt \*p int\* p int \* p**

**## 2 “ \*, & ” OPERATÖRÜ**

○ Bir değişkenin önüne konularak kullanıldığında (&var) bu değişkenin adresini temsil eder

○ Bir pointer’ın temsil ettiği değere ulaşılmak istendiğinde pointer isminin önüne konularak (\*pt) bu değere ulaşılır (dereference)

**## 3 POİNTER YAZMAK**

int \* p;

int number=5;

p=&number;

printf("%d",\*p);

int \*c=&number;

printf("%d",\*c);

**## 4 YAPILAN HATALAR**

int c;

int \*pc;

// pc bir adres fakat c degil

pc = c; // HATA

// &c bir adres fakat \*pc degil

\*pc = &c; // HATA

// &c ve pc adres

pc = &c;

// c ve \*pc deger

\*pc = c;

**## 5 POİNTER AND FUNCTİON**

Çünkü fonksiyonlar parametre olarak aldıkları değerleri önce kendi içlerinde oluşturdukları yerel değişkenlerine atayarak işleme sokarlar. Bu sebeple fonksiyon içerisinde aslında işlem gören değişkenler, fonksiyona parametre olarak gönderilen değişkenler değil fonksiyonun kendi yerel değişkenleridir**. (pass by value)**

**// in this one the value of a and will not change because it passed by value**

void swap\_by\_value(int a, int b)

{

int temp = a;

a = b;

b = temp;

}

Burada akla gelebilecek ilk soru bir değişkenin bir fonksiyon sonucu direkt olarak değerinin nasıl değiştirilebileceğidir. İşte bu noktada pointer’lar devreye girer. Eğer bir fonksiyon parametre olarak bir değişkenin değerini değil de değişkenin adresini alırsa, bu fonksiyonda bir yerel değişken oluşturulmaz, direkt adresi gönderilen değişken fonksiyon içerisinde işlem görür. Fonksiyon içerisinde işlem gören kısım bu adresin bulunduğu kısım olacaktır ve bu şekilde bir değişkenin değeri fonksiyon içerisinde değiştirilebilecektir. **(pass by reference)**

**// in this one the value of a and b will change because it passed by reference**

void swap\_by\_reference(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

**// şurada şöyle bir soru sorulabilir array ve stringleri fonksiyona yolluyorum ve değerlerini değiştirebiliyorum neden**

**// Bunun sebebi array ve stringlerin aslında pointer olmasından ötürüdür**

**KARAKTER DİZİLERİ STRİNGS**

String’ler esasında karakterlerden yani harflerden oluşan dizilerdir örneğin merhaba 7 karakterden oluşan bir kelimedir

char c[8] = {’m’, ’e’, ’r’, ’h’, ’a’, ’b’ ,’a’, ’\0’};(null terminator)

## 1 PRİNT THE STRİNGS

for(int i=0; c[i]!='\0';i++){

printf("%c",c[i]);

length++;

}

Or this one

printf("%s\n",k);

## 2 SCANNİNG STRİNGS

char string[100];

printf("Lutfen bir isim giriniz: ");

scanf("%s", string);

Kullanıcıdan girdi olarak bir string almak için formatlama olarak %s kullanılır fakat scanf içerisinde formatlamadan sonra string’e bir değer atamak için & işareti kullanılmaz. & işaretinin değişkenin adresini belirttiği önceki derslerde gösterilmişti. Bu durumda string’ler esasında pointer’lar işlevinde kullanılabilen yapılar oldukları için bir değer atanırken & kullanılmasına gerek yoktur.

Scanf just scan the part until the blank part after that it is not saved, so to solve this problem we need to use fgets(), puts()

## 4 SCANNİNG STRİNGS WİTH FGETS, PUTS string scanlama

**char string[100];**

**printf("Lutfen bir isim giriniz: ");**

**fgets(string, sizeof(string), stdin);**

**puts(string);**

Fgets((char\* var, sizeof(char \* var),stdin)

Puts(char \*) printf() gibidir ama sadece stringler için kullanılır ve yazdırdığı string’in sonuna otamatik olarak \n koyar.

## 5 STRİNG KÜTÜPHANESİ

## 6 STRİNG LENGTH WİTHOUT STRİNG (STRİNGİN UZUNLUĞUNU BULMA STRİNG KÜTÜPHANESİNİ KULLANMADAN)

strlen(char\* variable); // bir stringin uzunluğunu hesaplar

int stringlength(char string[1000]){

int length;

for(int i=0; string[i]!='\0'; i++)

{

length++;

}

return length;

}

strcpy(char\* destination, const char\* source); // Bir string’i başka bir stringe kopyalar

## 7 STRİNG COPY (STRİNG KOPYALAMA) strcpy

**Not:** String`ler birbirine direkt = yoluyla atanamaz. Bu noktada strcpy() fonksiyonunun kullanımı çok büyük önem taşır

1. WİTH STRİNG LİBRARY (STRİNG KÜTÜPHANESİNİ KULLANARAK)

Strcpy(char\* destination, const char\* source)

1. WİTHOUT STRİNG LİBRARY

void stringcopy(char string[SEN], char string\_1[SEN]){

int length= stringlength(string);

int length\_1=stringlength(string\_1);

for(int j=0; j<length\_1;j++){

string[j]=string\_1[j];

}

}

## 8 STRİNG APPEND (STRİNG EKLEME ARDINA) strcat

1. WİTH STRİNG LİBRARY (STRİNG KÜTÜPHANESİNİ KULLANARAK)

strcat(char\* destination, const char\* source)

1. WİTHOUT STRİNG LİBRARY

void stringcat(char string[SEN], const char string\_1[SEN]){

int length=stringlength(string);

int length\_1=stringlength(string\_1);

for(int i=length; i<length+length\_1-1;i++){

string[i]=string\_1[i-length];

}

}

## 9 STRİNG COMPERİSON (STRİNG KARŞILAŞTIRMA) strcmp

1. WİTH STRİNG LİBRARY (STRİNG KÜTÜPHANESİNİ KULLANARAK)

○ Eğer 0 döndürürse bu iki string aynıdır.

○ Eğer negatif bir sayı döndürürse str1 alfabetik olarak str2’den daha

önce geliyordur.

○ Eğer pozitif bir sayı döndürürse str2 alfabetik olarak str1’den daha önce geliyordur.

strcmp(const char\* destination, const char\* source)

1. WİTHOUT STRİNG LİBRARY

int stringcmp(const char string[SEN], const char string\_1[SEN]){

for(int i=0; string[i]!='\n';i++){

if(string[i]!=string\_1[i]){

return 1;

}

}

return 0;

}

**Static ve Dinamik Olarak Hafızada Yer Ayırma (Memory**

**Allocation)**

**STACK AND HEAP**

Kod içerisinde oluşturulan değişkenler işletim sisteminde belirli bir yer kaplar. Bu yerin boyutu kimi zaman belirli yani değiştirilemez iken kimi zaman ise kullanıcının program esnasında gireceği verilere göre değişebilecek durumdadır. Temel olarak bu farklılıktan dolayı verinin hafızada tutulduğu iki farklı bölge vardır. Verileri tutarken eğer program esnasında boyutları bildirilmiş değişmez bir yapı kullanılıyorsa **stack** bölgesinin, değiştirilebilir bir yapı kullanılıyorsa **heap** bölgesinin kullanılması gerekecektir. **Stack** **ve heap** kullanımları farklı ve dikkat edilmesi gereken bir konudur. **Stack** kullanılır, program sonlanana kadar boyutu değiştirilemez ve işi bittikten sonra kendini otomatik olarak bellekten yok eder. Fakat **heap**‘te program sırasında boyut değiştirilebilir ve program bitmeden kullanıcı tarafından bellekten yok edilmelidir çünkü bu işi kendisi yapmaz.

// stack bölgesini program san verir ve program bitince siler kendini

// heap i sen seçip ayarlarsın otomatik olarak bitmez senin sonlandırman gerekir

**Stack**:

● Oluşturulan değişkenler **stack** kapsamından çıkınca otomatik olarak yok edilir.

● Ulaşılması **heap**‘e göre oldukça hızlıdır.

● Stack üzerinde kullanım fazla olduğunda alan yeterli olmayabilir. Örneğin 20

boyutlu bir diziye 21 eleman atamak gibi.

● Oluşturulan değişkenler **pointer** olmadan kullanılabilir.

● Derleme zamanında oluşturulur.

● Kullanılacak yerin boyutu tam olarak biliniyorsa **Stack** kullanılmalıdır.

**Heap**:

● Bir blok içerisinde oluşturulan ve **heap’de** tutulan değişkenler, değişkenlerin

bulunduğu kod bloğunun dışına çıktığında otomatik olarak yok edilemezler,

bunun manuel olarak yapılması gerekir.

● **Stack** ile karşılaştırıldığında oldukça yavaştır.

● Doğru kullanılmaması durumunda bellek sorunları yaratır.

● Değişkenler **pointer** ile kullanılır.

● Çalışma zamanında oluşturulur.

3

● İhtiyaç duyulan boyut tam olarak bilinmiyorsa **Heap** kullanımı biçilmiş bir

kaftandır.

**STATİC MEMORY ALLOCATİON DYNAMİC MEMORY ALLOCATİON**

● **Statik** olarak hafızada yer ayırma **(Static Memory Allocation)**

○ Bu yöntemle değişkenlere kalıcı bir yer ayrılır.

○ Yer ayırma işlemi program başlamadan önce tamamlanmış olur.

○ Veriler **stack’de** depolanır.

○ Daha az verimli bir yöntemdir.

○ Bu yöntemde hafızanın yeniden kullanılabilirliği yoktur.

● **Dinamik** olarak hafızada yer ayırma **(Dynamic Memory Allocation**)

○ Bu yöntemle değişkenlere program aktif olduğunda yer ayırma işlemi

gerçekleştirilir.

○ Yer ayırma işlemi program çalışırken başlar ve sona erer.

○ Veriler **heap’te** depolanır.

○ Daha verimli bir yöntemdir.

○ Bu yöntemle hafıza tekrar tekrar kullanılabilir ya da serbest bırakılabilir.

**malloc()** dinamik olarak yer ayırır. Bir pointerin tutunabileceği veri boyutu dinamik olarak belirlenmiş olur

**realloc()** programın çalışması esnasında değiştimek içiin kullanılır

**free()** program sonunda hafızada ayrılmış yerin serbest bırakılmasını sağlar eğer bu yapılmazsa ram düzgün çalışmaz ve **leak** adı verilen problemler oluşur.

**malloc(), realloc(), free() KULLANIMI**

**malloc()**

veri tipi\* pointer=(veri tipi\*)malloc(size);

int\* ptr=(int\*)malloc(100\*sizeof(int));

// sizeof fonksiyonu veri tipi boyutunu döndürür

Sizeof(int)=4 float 4 double 8 char 1

**free()**

free(pointerın adı); // belleği serbest bırakır

// bu fonksiyon kullanıldıktan sonra bellek alanı serbest bırakılan pointer NULL pointera eşitlenmelidir .

// NULL değerine sahip bir fonksiyon hiçbir değer ifade etmez

**EXAMPLE**

int size;

printf("Dizinin buyuklugunu giriniz");

scanf("%d",&size);

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

printf("%d",toplam(ptr,size));

free(ptr); // free the malloc is necessary

ptr=NULL;

**realloc()**