YAPISAL PROGRAMLAMA

24.09.2019

şafak bilici

Switch Statements

```
switch (exp) {
   case const exp1: statement1;
      :
   case const expn: statementn;
   default: statementn+1;
}
```

```
switch (error_code) {
   case 1: printf("Gecersiz giris\n");
           break;
   case 2: printf("Gecersiz operand\n");
           break;
      :
   default: printf("bilinmeyen hata");
}
```

do while

```
do {
   ch = getch();
   if (ch == ' ') {
      num_of_space++;
   }
} while (ch != '\n');
```

for

```
for (j=10, i=0; i<10; i++, j--) {
   toplam = toplam + i
}
```

Arrays

```
int arr[5];
```

Element	Hex Adress
arr[0]	1000
arr[1]	1004
arr[2]	1008
arr[3]	100C
arr[4]	1010

```
int *ptr;
ptr = &arr[1]; ⟶ 1004
ptr = &arr[0]; ⟶ 1000
ptr = arr; ⟶ 1000
sizeof(ptr); ⟶ 4 byte
```

```
char *c;
char arr[5];
c = arr;
sizeof(c); ⟶ 4 byte
```

```
printf("the value: %d", arr[1]); → prints its value
printf("the adress: %p", &arr[1]); → prints its hex base adress
```

Dereferencing A Pointer

```
#include <stdio.h>
int main () {
    char *p-ch;                          // ch1 → 2001
    char ch1='A', ch2;                   // ch2 → 2002
    printf("address of p-ch is %p", &p-ch); // 1004
    p-ch = &ch1
    printf("value stored at p-ch is %p", p-ch); // 2001
    printf("the dereferenced value of p-ch is %c", *pch); // A
    ch2= *p-ch // ch2='A'
    return 0;
}
```

Variable	Address	Content
	1000	
p-ch	1004	2001

```
int j;
int *ptr_to_j = &j;
*ptr_to_j =10;
```

Ad	Var	Cont
1000	j	10
2000	ptr_to_j	1000

Pointer Arithmetic

Adress	Variable	Content
2000	p	1000

int *p;

p=p+3;

scaling

→ 3 objegit
3 tane 4 byte

→ 1000 + 4*3 = 100C

char *p;

p=p+3;

→ 1000 + 1*3 = 1003

→ 3 objegit
3 tane 1 byte

char a[4];

&a[3] - &a[0]

1003 1000

=3

Var	Address
a[0]	1000
a[1]	1001
a[2]	1002
a[3]	1003

→ 2 byte

Short a[4]

&a[3] - &a[0]

1006 1000 = 3

Accessing Array Elements Through Pointers

int *ptr;

int arr[5];

ptr= &arr[0]

→ arr[2] ⟷ *(ptr+2) (same)

Ad	Var	Content
1000	arr[0]	
1004	arr[1]	
1008	arr[2]	
⋮		
2000	ptr	1000

```
int t=1, *p_to_t;

*(&t)=2

*p_to_t=3
```

Ad	Var	Cont
1000	t	1 → 2 → 3
⋮		
2000	p_to_t	1000

~

```
float ar[5], *p;
p=ar; ⟶ ✓
ar=p; ⟶ X
&p=ar; ⟶ X
ar++; ⟶ X
ar[1]=*(p+3); ⟶ ✓
p++; ⟶ ✓
```

~

```
char arr[5]={'a','b','c','d','e'};
char *p;
p=&arr[3];
printf("%c",*(p+1)); ⟶ prints 'e'
```

~

```
int arr[5]={1,2,3,4,5};
int *p1, *p2;
int j;
char *p3;
printf("fark %d", &arr[3]-&arr[0]); //3
p1=a;
p2=p1+4;
j=p2-p1;
printf("j:%d",j); //4
j=p1-p2;
printf("j:%d",j); //-4
p1=p2-2;
printf("%d",*p1); 4
```

a[0] ⟶ 1000
p1 ⟶ 2000
p2 ⟶ 2004

Pointers to Pointers

```
int a=20;
int *b = &a;
int **c= &b;
```

Var	Address	Content
a	1000	20
b	1008	1000
c	1010	1008

→ a=20
*b=20
**c=20

```
char arr[5] = {'a', 'b', 'c', 'd', 'e'}
char *p1-ch, **p2-ch;
int i;
p1-ch= arr;        → &arr[0]
*(p1-ch+3) = '*';
p2-ch = &p1-ch;
**p2-ch = 'z';
```

1000 → arr[0] → a → z
1001 → arr[1] → b
1002 → arr[2] → c
1003 → arr[3] → d → *
1004 → arr[4] → e

2000 → p1-ch → 1000
2008 → p2-ch → 2000

Null Pointer

```
char *p;
```

p=0; ~~ p=NULL; ~~ char *p= 0; ~~ char *p=NULL;

```
int main()
{
    int *i, *j;
    int *ii=0, *jj=0;

    if (i==j) {
        printf("i and j are same by chance");
    }
  if(ii==jj) {
        printf("ii and jj are always same");
  }
```

int toplama Rul (int *DizList, int n);

Arrays of Pointers

```
char *ar_of_p[5];
char c0 = 'a';
char c1 = 'b';
ar_of_p[0] = &c0;
ar_of_p[1] = &c1;
```

Element	Adress	Content
ar_of_p[0]	1000	2000
ar_of_p[1]	1004	2001
ar_of_p[2]	1008	
ar_of_p[3]	100C	
ar_of_p[4]	1010	

c0 ⟶ 2000 ⟶ a

c1 ⟶ 2001 ⟶ b

Strings

```
char Kelime[] = "some";
```

Element	Address	Content
Kelime[0]	2010	's'
Kelime[1]	2011	'o'
Kelime[2]	2012	'm'
Kelime[3]	2013	'e'
Kelime[4]	2014	'\0' → NULL

01.10.2019

```
int a[4] = {1, 2, 3, 4}, *b, *c;
```

c = a;

b = &(a[3]);

*c = *c * 2;

c = b;

*b = 5;

b--;

*b = *c

Multidimensional

int mtr[10][10];

unsigned char image[100][100][3];

```
int exp[5][5] = { {1,2,3},
                  {4},
                  {5,6,7} }
```

prints

1 2 3
4 0 0
5 6 7
0 0 0
0 0 0

int exp[][2] = {{1,2},{3,4}}

int arr[2][3] = {{0,1,2},{3,4,5}}

```
Short ar[4];
Short *p;
p=ar; ✓
ar=p; X
&p=ar; X
ar++; X
p++; ✓
ar[1] = *(p+3); ✓
```

Element	Adress	Cont
arr[0][0]	62FE10	0 ← arr / arr[0] / &arr[0][0]
arr[0][1]	62FE14	1
arr[0][2]	62FE18	2
arr[1][0]	62FE1C	3 ← arr[1] / &arr[1][0]
arr[1][1]	62FE20	4
arr[1][2]	62FE24	5

arr[1][2] <==> *(arr[1] + 2) <==> *(*(arr+1)+2)

* strcmp (*char, *char) ⟶ compares strings

```
char *dict[LAST_LANG][MAX_WORDS] = {{"abhor", "able", "abort", NULL},
                                    {"absurde", "accep", "accord", "achat", NULL}};
```

↓

char pointer yaparak string matrisi yapabildik. Pointer olmasa char matrisi olurdu.

Fonksiyonlar

```
int topla(int a, int b){
    return a+b;
}
```

⟶ call by value

```
main()
sonuc = topla(a, b);
```

```
Void swap(int *a, int *b){
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

⟶ call by reference

```
main()
swap(&a, &b);
```

```
int main() {

    int arr[20], n, i, toplam;
    int *arr2;
    printf("Dizinin elaman sayisi: ");
    scanf("%d", &n);
    for(i=0; i<n, i++){
        scanf("%d", &arr[i]);
    }
    toplam = topladiziV1(dizi,n);
    toplam = topladiziV2(dizi,n);
```

```
int topladiziV1 (int *dizi, int boyut){
    int i, toplam=0;
    for(i=0; i<boyut; i++){
        toplam += dizi[i];
    return toplam;
}

int topladiziV2 (int dizi[], int boyut){
    int i, toplam=0;
    for(i=0; i< boyut; i++){
        toplam += dizi[i]; // toplam += *(dizi+i)
    }
    return toplam;
}
```

Matrix to Function

08.10.2019

```
int main() {
    int mat[2][3] = {{0,1,2},{3,4,5}};
    printf("toplam : %d", toplaMat1(mat,2,3));
    printf("toplam : %d", toplaMat2(mat,2,3));
    int mat2[50][50] = {{0,1,2},{3,4,5}};
    printf("toplam : %d", toplaMat2(mat2,2,3));
    printf("toplam : %d", toplaMat3(mat2,2,3,50));
int toplamMat1(int **mat, int dim1, int dim2) {
    int sum=0, i, j;
    for(i=0; i<dim1; i++) {
        for(j=0; j<dim2; j++) {
            sum = sum + *((int*)mat + i*dim2 + j); }}
    return sum; }
int toplamMat2(int mat[][50], int dim1, int dim2) {
    int sum=0;
    for(i=0; i<dim1; i++) {
        for(j=0; j<dim2; j++) {
            sum = sum + mat[i][j];
        }
    }
    return sum;
}
int topkem Mat3(int **mat, int dim1, int dim2, int col) {
    int sum=0, i, j;
    for(i=0; i<dim1; i++) {
        for(j=0; j<dim2; j++) {
            sum += *((int*)mat + i*col + j);
    } }
    return sum;
}
```

(*(*(mat+i)+j))

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	10	11	12	13	14	15	-			
2	20	21								
3	30	31								

i=1 j=2

1*10+2=12

Dynamic Memory Allocation

malloc();
calloc();
realloc();
free();

Syntax

```
int n=5;
int *dizi;          → yer ayırmaz
dizi = (int *) calloc (n, sizeof(int));
                       n: adet, sizeof(int): obje boyutu
dizi = (int *) malloc (n*sizeof(int));
int *dizi2;
dizi2 = realloc (dizi, (n+3)*sizeof(int));
                       (n+3): new size
```

→ dizi olsaydı eski bilgileri kaybedebilirdik.

```
int main() {
    int *OgrNotDizi;
    int n,m;
    float ort;
    scanf("%d", &n); //öğrenci sayısı
    OgrNotDizi = (int*) calloc (n, sizeof(int));
    OgrenciAl = (OgrNotDizi, 0, n);
    //OgrNotDizi = yerac(n);
    ort = OrtalamaBul (OgrNotDizi, n);
    scanf("%d", &m);
    OgrNotDizi2 = YerGenislet (OgrNotDizi, n, m);
    //YerGenisletV2 (OgrNotDizi, OgrNotDizi2, n, m);
    if (OgrNotDizi2 == NULL) {
        printf("yer genisletilemedi");
        exit(0);
    }
    OgrenciAl (OgrNotDizi2, n, m);
```

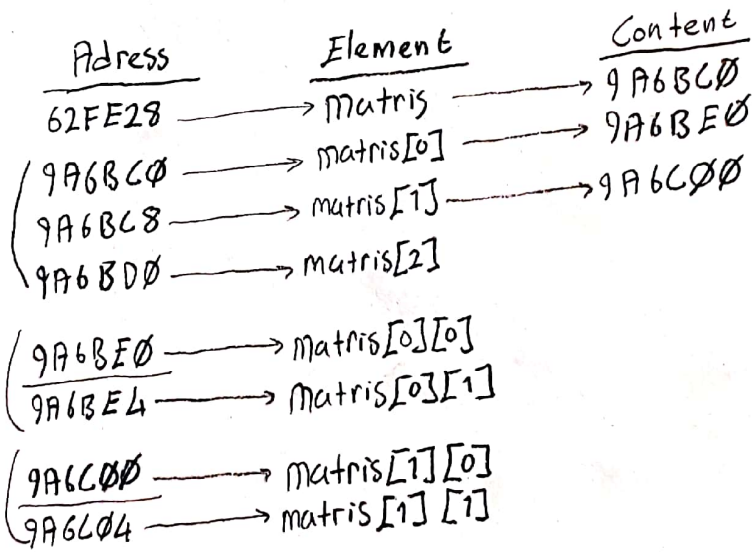
```
int *yerac(int n) {
    int *dizi;
    dizi = (int*) calloc(n, sizeof(int));
    return dizi;
}

Void OgrenciAl (int *OgrList, int basla,
                int sayı)
{
    int i;
    for (i=basla; i<basla+sayı; i++) {
        scanf("%d", &OgrList[i]);
    }
}
```

```
float OrtalamaBul(int *OgrList, int n){
    int c, sum=0;
    for(c=0; c<n; c++){
        sum += *(OgrList+c); // OgrList[c]
    }
    return (float) toplam/n;
}

int *YerGenislet(int *OgrList, int n, int m){
    int *OgrListNew;
    OgrListNew = realloc(OgrList, (m+n) * sizeof(int));
    return OgrListNew;
}

Void YerGenisletV2(int *OgrList, int **OgrList2, int n, int m){
    *(OgrList2) = realloc(OgrList, (m+n) * sizeof(int));
}
```

15.10.2019

Matrix Allocation

```
int row,col;
scanf("%d", &row, &col);
float **matris;
matris=(float**) calloc (row, sizeof(float*))
for (i=0; i<row; i++) {
    matris[i] =(float*) calloc (col, sizeof(float));
}
```



15.10.2019

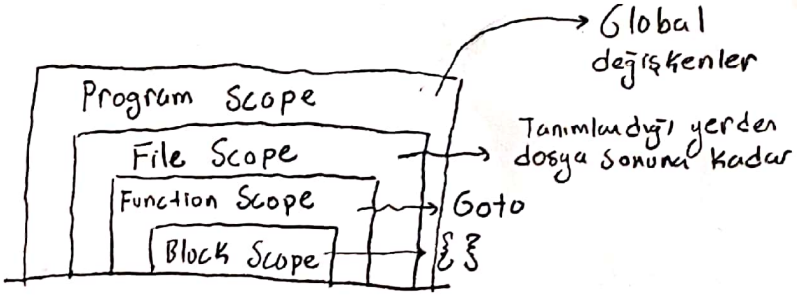
Scope / Duration

Scope ⟶ değişkenin aktif olduğu kod parçası

Duration ⟶ Memory'de saklanma süresi

- Automatic
- Fixed
  - fonksiyon içinde
  - statik

HEAP	⟶ Dynamic Memory Allocation
STACK	⟶ Local Variables
GLOBAL	⟶ Global Variables
CODE	⟶ Machine Code




```
#include<stdio.h>
int c=15; // program scope
static int j=3; // file scope
Void funcA();
Void printfc();
Void fixDuration();
int main() {
    int k=5, c; // Block scope
    for(c=0; c<3; c++) {
        k++;
        goto PrintAgain;
    }
    PrintAgain:                          → Block Scope c, global olan etkilenmez
    printf("%d %d %d", c, j, k); // 0, 3, 6
```

①

```
Void fixDuration(){
    Static int fix=2; // Block scope fix duration

    fix++;
    printf("fix: %d", fix);
```

```
Void funcA(){
    int k=10, c; // block scope
    for(c=0; c<5; c++){
        int a=4; //block scope
        printf("%d %d %d", k, a, c);
        printfc(); //15
    }
    if(a==4){
        printf("esit");
    }
}
Void printfc(){
    printf("%d", c);
}
```

iterasyon ↑

//10,4,0 ↑

compiler hatası a, block scope

①

```
funcA();
fixDuration(); //3
fixDuration(); //4
}
```

Definations and Allusions

For Global Variables

defination: ⟶ variable için memory de yer ayırır

allusion: ⟶ başka bir dosyada define edilmiş bir global değişkeni kullanırken

```
Void func(){
    extern int i;
    extern float arrayf[5];
}
```

```
int j=0; // global definition
extern float x=1.0; // global definition
void funch(){
    int k=0; // local variable
    extern int i; // allusion to a global variable
}
```

Storage Classes

Auto: Default

Static: fonksiyon içindeyse fix duration, dışında file scope

Extern: fonksiyon içindeyse global allusion, dışında global definition

Const: degeri değiştirilemez

Register:

```
int strlen (register char *p){
    register int len=0;
    while (*(p++))
        len++;
    return len;
}
```

Volatile: optimizasyonu kapatır.

```
Void get_two_kbd_chars (){
    extern volatile char KEYBOARD;
    char c1, c2;
    c1=KEYBOARD;
    c2= KEYBOARD;
}
```

Recursion

```
int main(){
    void recurse();
    recurse();
    return 0;
}
```

```
void recurse(){
    static int count=1;
    printf("%d", count);
    count++;
    recurse();
}
```

Stack dolma kadar

```
Void recurse(){
    Static int count=1;
    if(count>3) return;
    else{
        printf("%d", count);
        count++;
    }
    recurse();
}
```

sonlu

```
Void recurse2(int count){
    if(count>3) return;
    else{
        printf("%d", count);
        count++;
        recurse(count);
    }
}
```

```
int factorial(int n) {
    if(n==0){
      return 1;
    }
    else{
        return n* fact(n-1);
    }
```

```
int main(){
    int num=4;
    printf("%d", factorial(num));
    return 0;
}
```

Structs

```
Struct{
        char ogr_isim[15], ogr_soyisim[20];
        short ogr_yas;
        float ogr_puan;
} ogrenci1;  // ogrenci1 = {"irem", "turkmen", 20, 100}
```

(1) without tag

~ ~

```
Struct ogr {
        char ogr_isim[15], ogr_soyisim[20];
        short ogr_yas;
        float ogr_puan;
} ogrenci1, ogrenci2, ogrenci3[10];
```

(2) with tag

~ ~

```
Struct ogr {
        char ogr_isim[15], ogr_soyisim[20];
        short ogr_yas;
        float ogr_puan;
};

Struct ogr ogrenci1;
```

(3) with tag

~ ~

```
typedef struct {
        char ogr_isim[15], ogr_soyisim[20];
        short ogr_yas
        float ogr_puan;
} ogr;

ogr ogrenci1;
```

(4) typedef

Referencing Structure Members

ogrenci

ogr. ogr_isim

ogrenci

(*ogr). ogr_isim

ogrenci ⟶ ogr_isim

STRUCTS

22.10.2019

```
struct ogr {

    char ogr_isim[15], ogr_soyisim[15];
    short ogr_yas;
    float ogr_puan;
};

int i, max=0, maxyer;
struct ogr *ogrenci, ogrenciler[3];
struct ogr *maxyerstr;

for(i=0; i<3; i++){
    scanf("%s %s %d %f", ogrenciler[i].ogr_isim, ogrenciler[i].ogr_soyisim,
                         &ogrenciler[i].ogr_yas, &ogrenciler[i].ogr_puan);
    if(ogrenciler[i].ogr_puan > max){
        max = ogrenciler[i].ogr_puan;
        maxyer = i;
    }
}
printf("%s %d", ogrenciler[maxyer].ogr_isim, max);
ogrenci = &ogrenciler[maxyer];
printf("%s", ogrenci -> ogr_soyisim); // printf("%s", (*ogrenci).ogr_soyisim);

ogrenci = ogrenciler;
max = 0;
for(i=0; i<3; i++){
    if(ogrenci->ogr_puan > max){
        max = ogrenci -> ogr_puan
        maxyerstr = ogrenci;
    }
    ogrenci++;
}
printf("%s ", maxyerstr -> ogr_soyisim
```

(->) adres üzerinden ulaşmak için

Dynamic Memory Allocation
for Structures

```
typedef struct {
        char ogr_isim[15], ogr_soyisim[20];
        short ogr_yas;
        float ogr_puan;
} ogr;

int main() {
    ogr *ogrenciler;
    int n, i;
    scanf("%d", &n);
    ogrenciler = (ogr *) calloc (n, sizeof(ogr))
    ogrenciler[0].ogr_yas = 19; // ogrenciler -> ogr_yas = 19; or (*ogrenciler).ogr_yas = 19;
    free (ogrenciler);
```

Nested Structures

```
typedef struct {
        char gun;
        char ay;
        short yll;
} TARIH;

typedef struct {
        char isim[30];
        int maas;
        TARIH dtarih;
} CALISAN;

int main() {
    CALISAN calisan2 = {"ali", 17000, {11, 10, 90}};
    CALISAN calisan1;
    calisan1.dtarih.ay = 2;
    calisan1.maas = 10000;
    calisan1.dtarih.gun = 20;
    return 0;
}
```

```
typedef struct {
        char isim[30];
        int maas;
        struct {
            char gun;
            char ay;
            short yel;
        } tarih;
} CALISAN;
```

Passing Structures as Function Arguments & Returning Structures

```
#include<stdio.h>

typedef struct {
        char isim[30];
        short yas;
        float puan;
}OGR;

int main() {
   OGR ogrenciler[3], ogrenci1, ogrenci2, *ogrptr;
   int i, n=3;
   for(i=0; i<n; i++) {
      scanf("%s %d %f", ogrenciler[i].isim, &ogrenciler[i].yas, &ogrenciler[i].puan);
   }
   Findmax(ogrenciler, n);
   scanf("%s %d %f", ogrenci2.isim, &ogrenci2.yas, &ogrenci2.puan);
   notguncelle(&ogrenci2);
   ogrenci3 = notguncelleV2(ogrenci2);
   ogrptr = yerAc(3);
   return 0;
}
```

```
OGR * yerAc(int n){
    return (OGR*) calloc(n,sizeof(OGR));
}
```

```
Void Findmax(OGR *ogrenciDizi, int n){
    int i, max=0;
    for(i=0; i<n; i++){
        if(ogrencidizi[i].puan > max){
            max = ogrencidizi[i].puan;
        }
    }
    printf("%d", max);
}
```

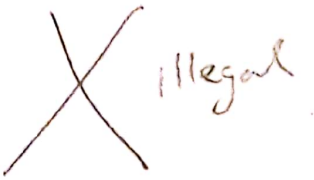
```
Void notguncelle(OGR *ogrenci){
    if(*(ogrenci).yas<18){
        (ogrenci->puan)++;
}
```

```
OGR *notguncelleV2(OGR ogrenci){
    static OGR *ogrenci1;
    if(ogrenci.yas < 18)
      (ogrenci.puan)++;
    ogrenci1 = &ogrenci;
    return ogrenci1;
}
```

Self Referencing Structures

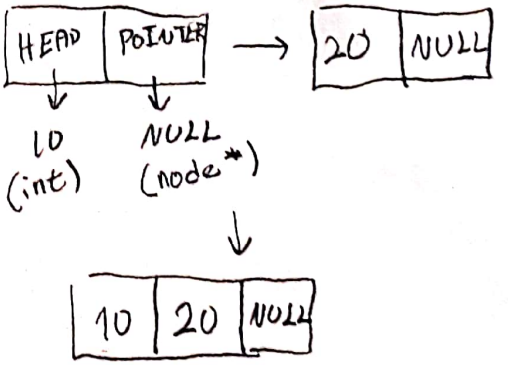
```
struct s {
    int a, b;
    float c;
    struct s *pointer_to_s;
};
```

```
typedef sr {
    int a, b;
    float c;
    sr * pointer_to_sr;
} sr;
```




```
int main () {
    struct s s1 = { 1, 2, 1.2, NULL };
    struct s s2 = { 3, 4, 5.6, NULL };
    s1.pointer_to_s = &s2;
    s2.pointer_to_s = &s1;
    printf("%d %d", (s1.pointer_to_s)->a, (s1.pointer_to_s)->b); // 3, 4
    printf("%d %d", (s2.pointer_to_s)->a, (*s2.pointer_to_s).b); // 1, 2
```

```
struct node {
    int val;
    struct node *next;
};
int main () {
    struct node *head, *node2;
    int num;
    head = (struct node *) malloc (sizeof(struct node));
    num = 10;
    head -> val = num;
    head -> next = NULL;
    node2 = (struct node*) malloc (sizeof(struct node));
    node2 -> val = 20;
    node2 -> next = NULL;
    head -> next = node2;
    list (head);
```



```
Void list (struct node * head) {

    struct node *current = head;
    while (current -> next != NULL) {
        printf("%d ", current -> val);
        current = current -> next;
    }
    printf("%d", current -> val);
}
```