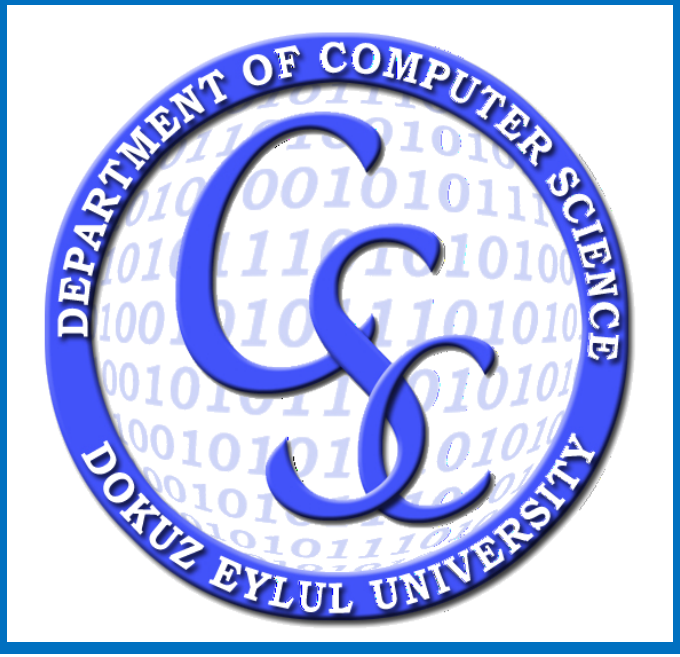


BİL2001
Algoritmalar ve Veri Yapıları

Öğr. Gör. Dr. Alper VAHAPLAR
2019 – 2020

Bazı Tanımlar

- Veri Tipi (Data Type): Bir değişkenin alabileceği değerler kümesini ve o değişken için ayrılacak bellek miktarını belirleyen ifade.
- Veri Yapıları (Data Structures): Bilgisayar ortamında verilerin etkin olarak saklanması ve işlenmesi için kullanılan yapılardır.
- Veri Modeli (Data Model): Belirli bir veri kümesinin mantıksal ya da dizisel durumu.

Veri Yapısı – Veri Modeli

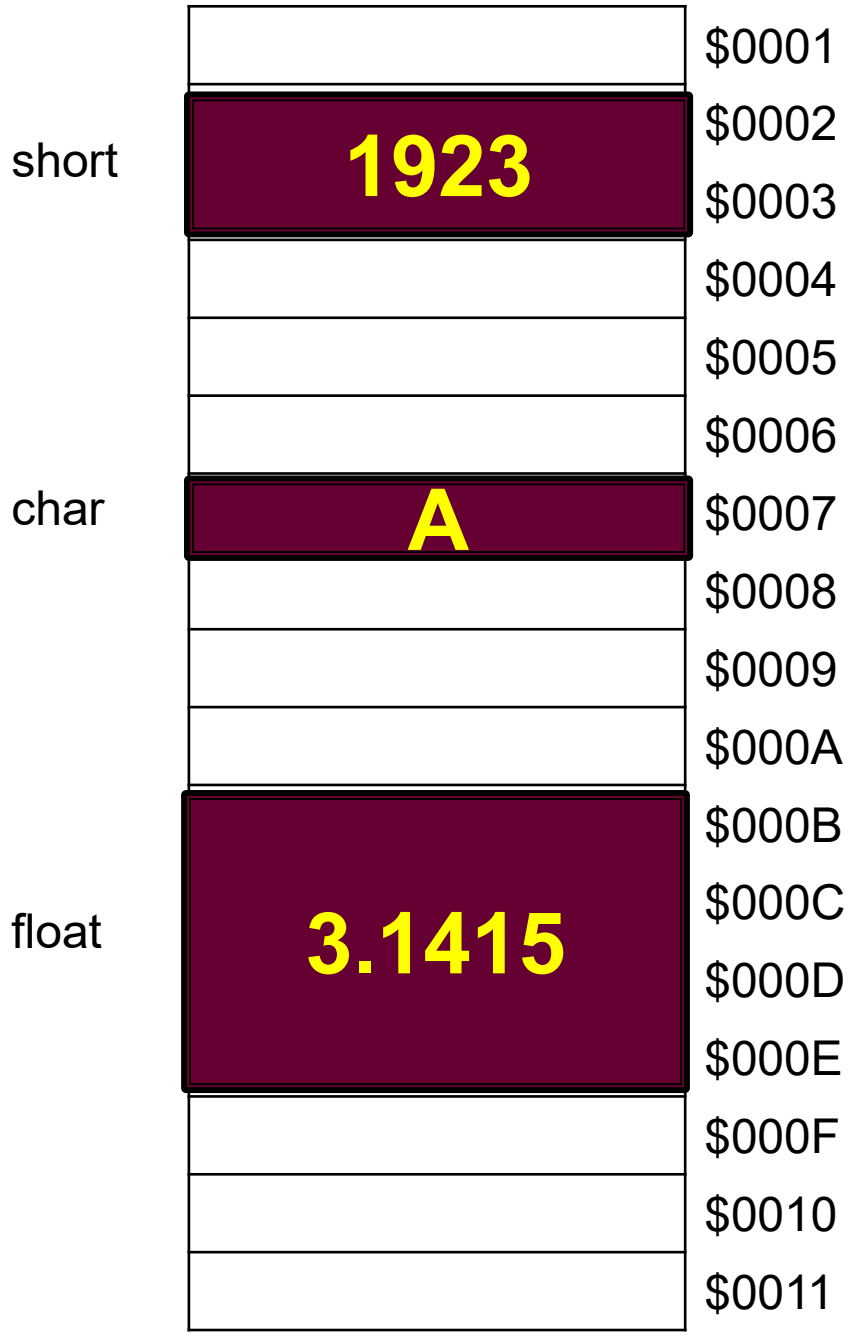
- Veri yapısı, bilginin saklanış biçimini ifade eder.
- Her problem dizilerle (array) çözülebilir,
- ancak etkin olmaz.
- Ör: *Motorun yapısını, şanzımanın çalışma mekanizmasını bilmeden de araba kullanılabilir.*
- Bilgisayar Bilimci/Mühendisi – programcı farkı.

Veri Modeli

- Problemin çözümü için kavramsal bir yaklaşım yöntemi
- Veriler arasındaki ilişki ve bağlantılarla ilgilenir,
- Veriler üzerinde işlem yapacak olan algoritmalar veri modeline göre ifade edilir.

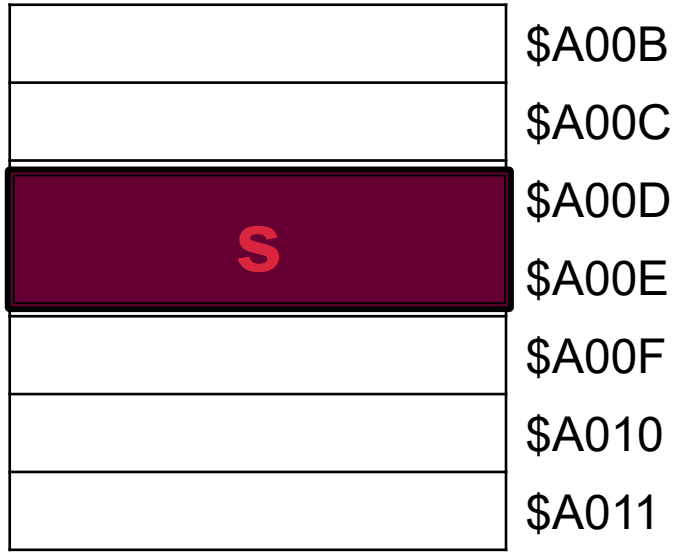
İşaretçiler (Pointers)

- Bir değişkenin, bir verinin ya da bir program parçasının o anda bellekte işgal ettiği bellek adresini tutan değişkendir.
- Verinin kendisi değil, bellek adresinin başlangıç değeri saklanır.
- İşaretçiler bellekte ne kadar yer kaplar?
  - Mikroişlemcinin adres uzunluğu kadar (doğrusal adresleme)
  - 32 bit veya 64 bit




```
main( )
{
    short tam=1923;
    char harf='A';
    float kesirli=3.1415;

    short *s;char *c; float *f;
}
```



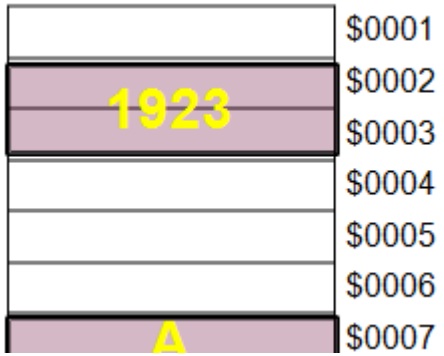
İşaretçiler – Pointers

```
main( )
{
   short tam=1923;char harf='A';float kesirli=3.1415;

   short *s;char *c; float *f;
   printf ("s= %d\n",s);  // Adres bilgisi görünür
   s=&tam; //işaretçinin değerine "tam" değişkeninin adresi atanıyor.(0002)
   printf ("tam'ın adresi = %d\n",s); // "tam"ın adres değeri
   printf ("O adresteki değer=%d\n",*s); // "s"nin işaret ettiği yerdeki değer

   s++;   // s=s+1 ???
   printf ("s = %d\n",s); // s'nin değeri kaç artmış?
   printf ("*s'nin boyutu = %d\n",sizeof(*s)); // s'nin işaret ettiği yer
   printf ("s'nin boyutu (adres olarak) = %d\n",sizeof(s)); //s'nin kapladığı

}
```



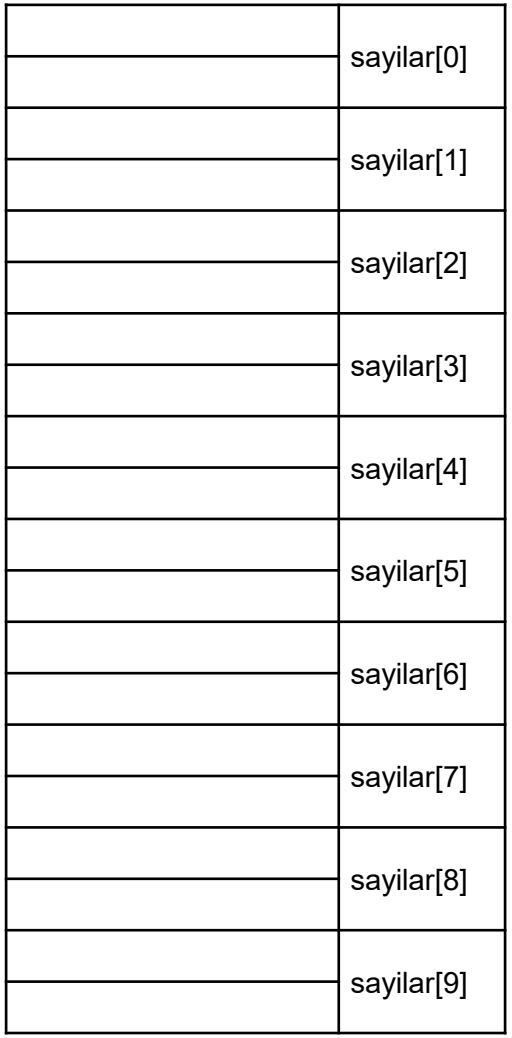
Diziler – Arrays

- Aynı türden verileri tek bir isim altında saklamak için kullanılır.
- Static diziler, program sırasında sabit boyutta elemana sahiptir.
- Çok boyutlu olabilirler.
- İlk eleman 0 ile başlar.
- Son eleman (boyut – 1)nci elemandır. (C?)

Diziler – Arrays

- Ör:

  short sayilar[10]



Diziler – Arrays

- Dizinin adı aynı zamanda işaretçidir.

```
main( )
{
  short sayilar[10];
  short *p;
  sayilar[3]=123;
  p=sayilar; //& işaretine gerek yok;
  printf ("p = %d\n",sayilar);
  printf ("4. Eleman = %d\n",*p+3)); // ???
  printf ("4. Eleman = %d\n",*(p+3));
}
```

Diziler – Arrays

- İki boyutlu diziler

```
main( )
{
  short sayilar[10][20];
  short *p;
  sayilar[3][5]=456;
  p=*sayilar; //Bu sefer “*” gerek...
  printf ("p = %d\n",sayilar);
  //sayilar[3][5]’i p cinsinden ifade edin
  printf ("p = %d\n",*(p+(3*20+5)));
}
```

- short A[m][n]
- A[i][j] = *(p+(i*n+j))

Veri Modelleri

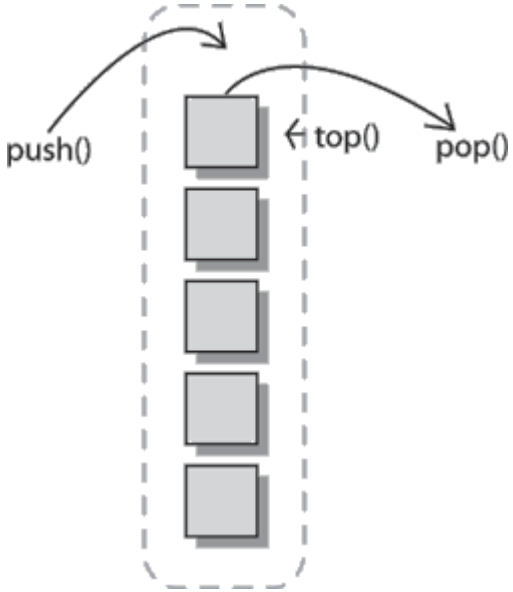
- Uygulama için en etkin programın yazılmasını sağlar,
- Yazılacak olan programın çalışma hızı ve bellek gereksinimi hakkında bilgi verir,

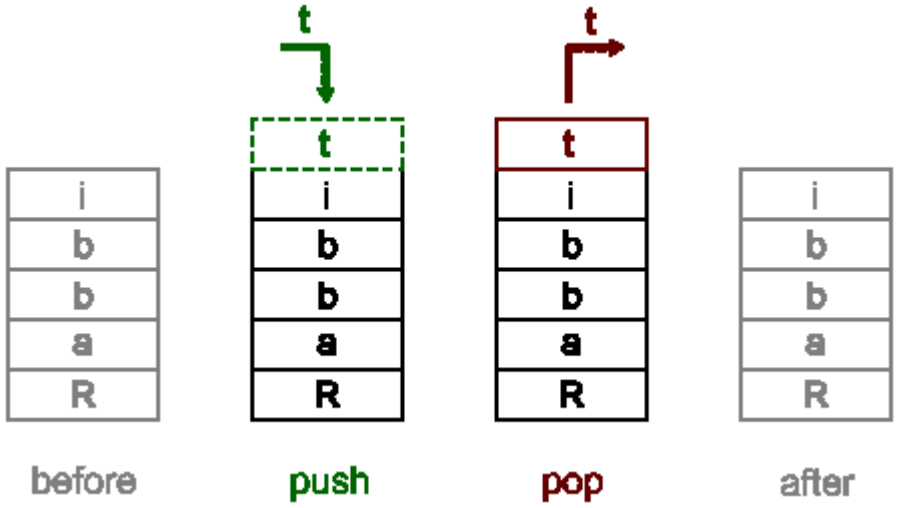
Veri Modelleri

- Stack (Yığın)
- Kuyruk (Queue)
- Linked List (Bağlı/Bağlantılı Liste)
- Tree (Ağaç)
- Graph (Çizge / Graf)
- State Machine (Durum Makinası)

Veri Modelleri

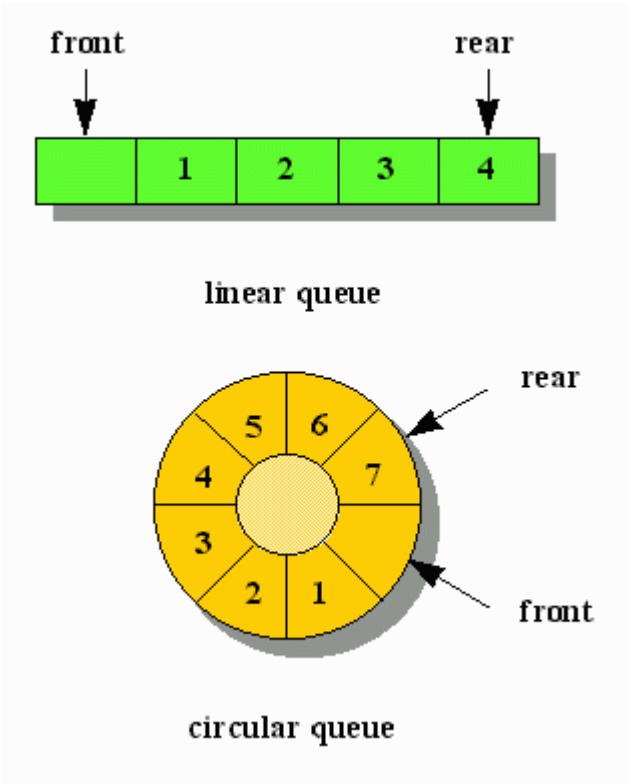
- Stack (Yığın) Veri Modeli





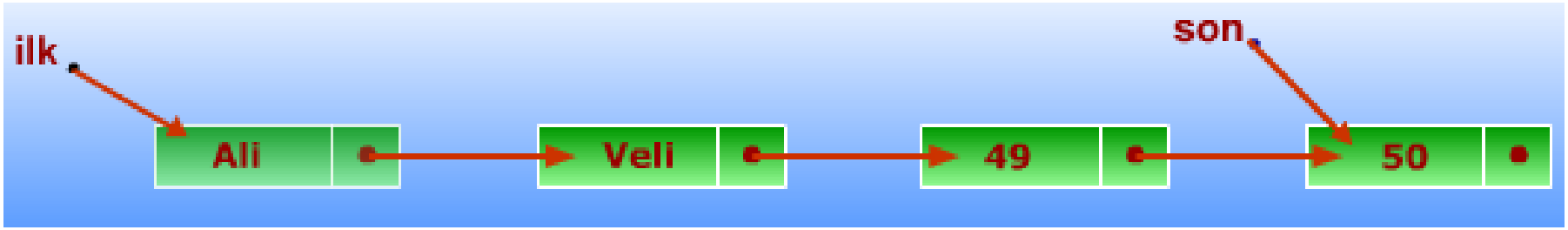
Veri Modelleri

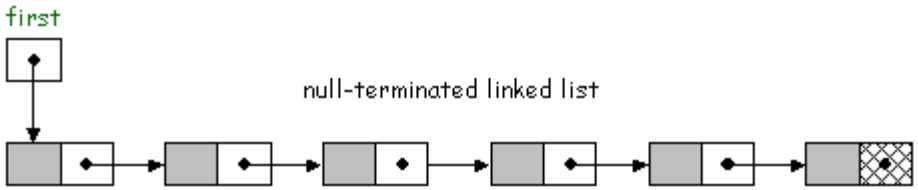
- Queue (Kuyruk) Veri Modeli

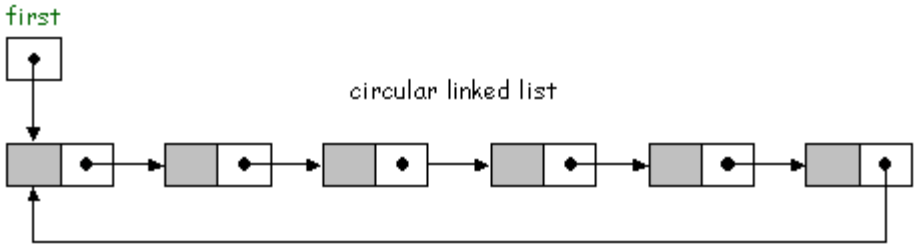


Veri Modelleri

- Linked List (Bağlı Liste) Veri Modeli

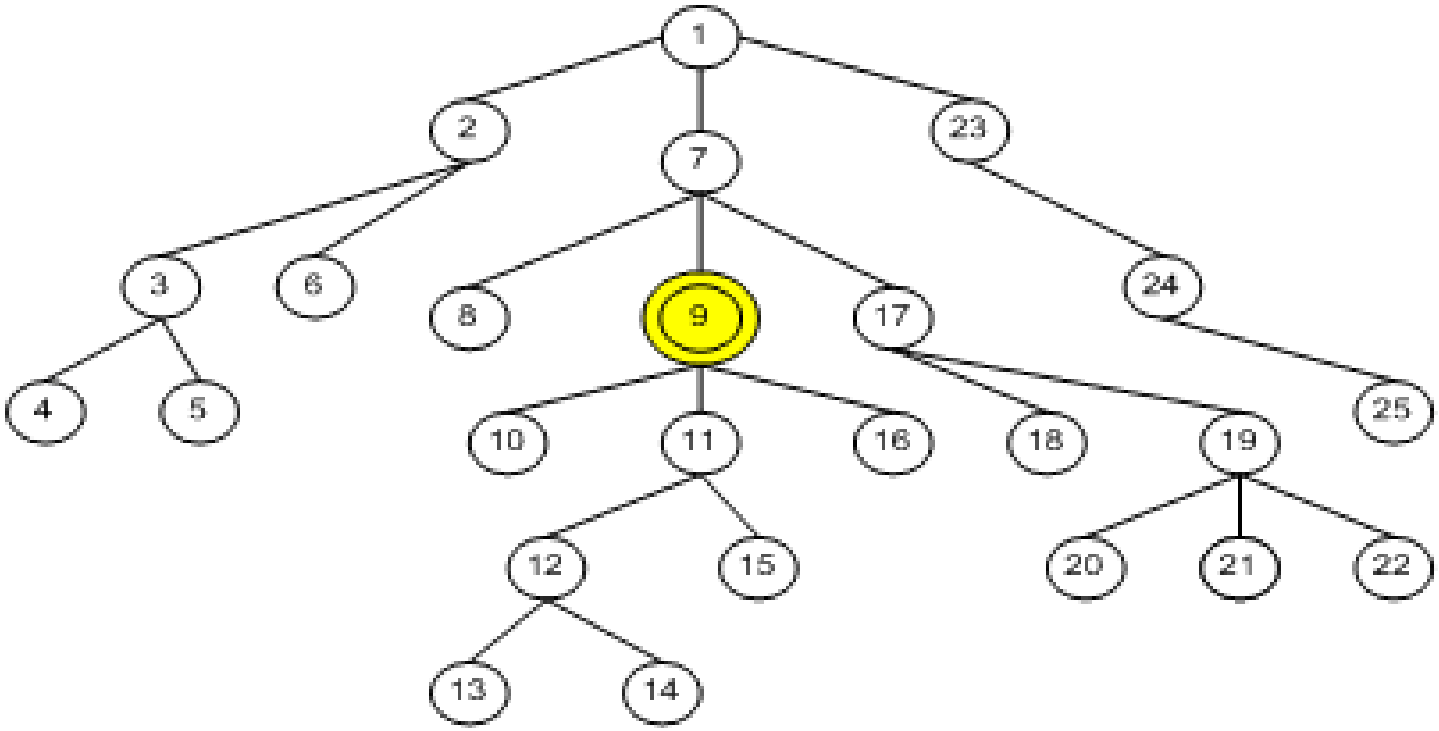






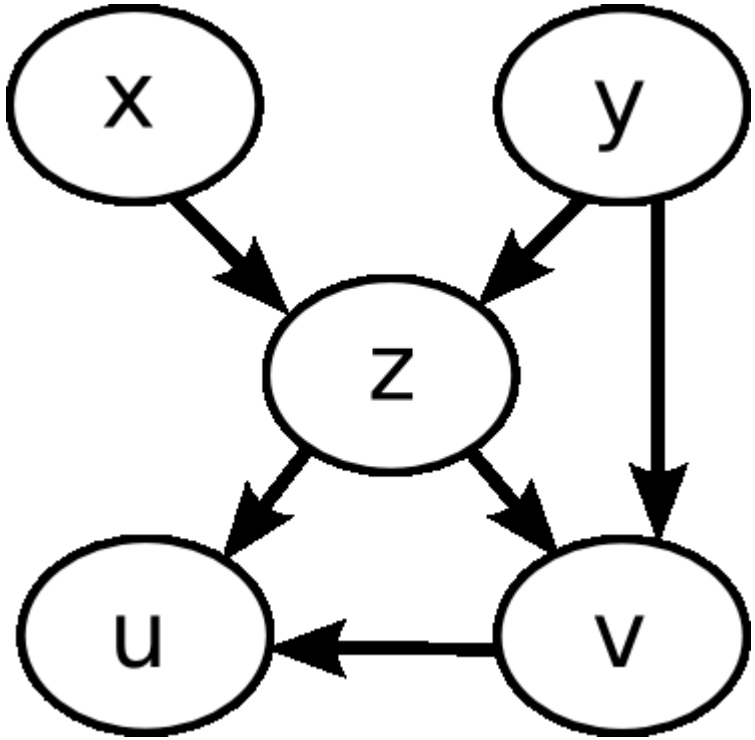
Veri Modelleri

- Tree (Ağaç) Veri Modeli



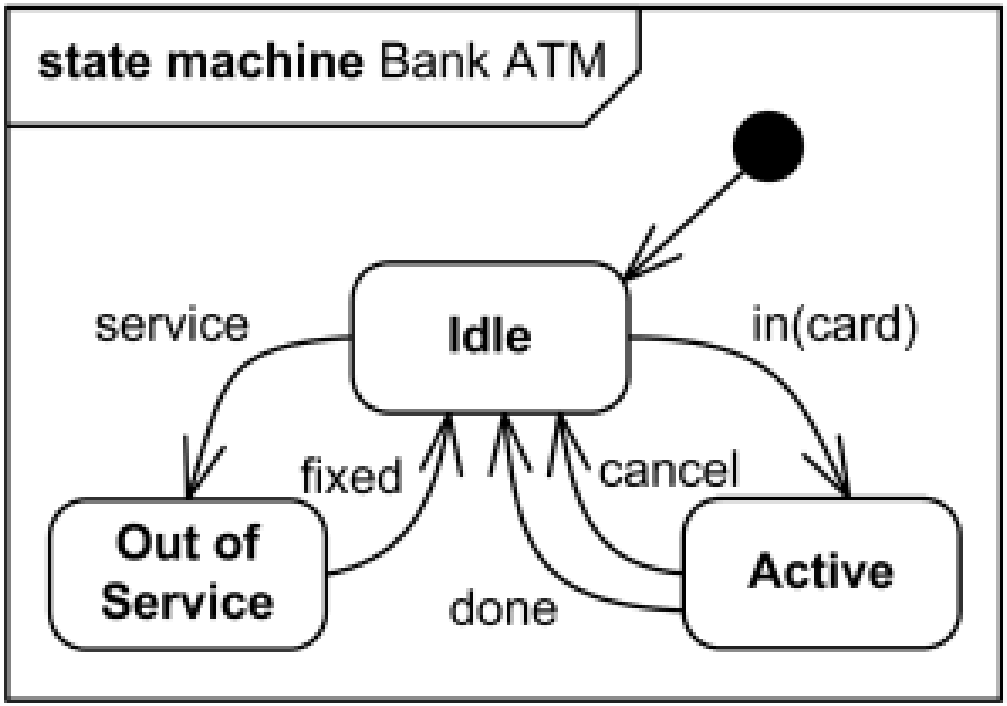
Veri Modelleri

- Graph (Çizge) Veri Modeli



Veri Modelleri

- State Machine (Durum Makinası) Veri Modeli



Stacks – Yığınlar

- Bellek adresleme üzerine kurulu,
- Sıralama ve geçici saklama için kullanılan,
- LIFO (Last In First Out) Son giren ilk çıkar mantığıyla çalışan,
- Soyut (abstract) veri yapısıdır.
- Temel iki işlem:
  - push (en üste veri ekleme)
  - pop (en üstten veri alma ve çıkarma)

Stacks – Yığınlar

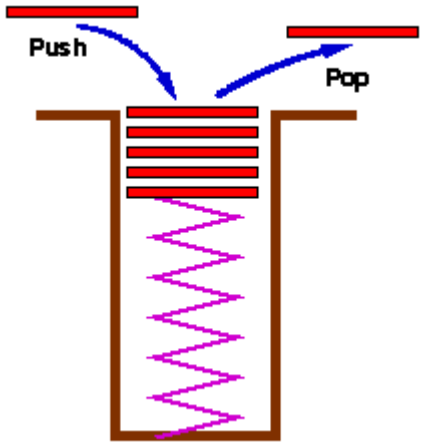
- Yığın Örnekleri

Stacks – Yığınlar

- Array ile stack uygulaması

```
int stack[10]={0}, top=0;
main()
{
   push(5);
   push(9);
   push(7);
   pop();
   pop();
   push(12);
}
```

5	stack[0]
12	stack[1]
	stack[2]
	stack[3]
	stack[4]
	stack[5]
	stack[6]
	…



Alıştırma

- Array kullanarak stack yapısını gerçekleştiren programı yazınız…

Stacks – Yığınlar

- PREFIX – INFIX – POSTFIX gösterim
- İşlemciler (Operators)'in Özellikleri
- Arity (Number of Operands)
  - Unary Operators (– , NOT, &, *(pointer))
  - Binary (+, -, *, /, AND, OR, >, etc.)
- Precedence
  1. exponentiation
  2. - (negation)
  3. *, /
  4. DIV (integer division)
  5. MOD (modulus)
  6. +, -

Stacks – Yığınlar

- PREFIX – INFIX – POSTFIX gösterim
- İşlemciler (Operators)'in Özellikleri
- Associativity
  - Left Associatives
    - +, -, *, /
    - 4 – 2 – 1
  - Right Associatives
    - ^ (exponentiation)
    - 2^3^4 = 2^81

Stacks – Yığınlar

- PREFIX – INFIX – POSTFIX gösterim
- İşlemciler (Operators)'in Özellikleri
- Place of Operands (İşlenenlerin Yeri)
  - Infix (X+Y, A * (B + C) / D)
  - Postfix (X Y + , A B C + * D /)
  - Prefix (+ X Y, / * A + B C D)

Stacks – Yığınlar

Infix	Postfix	Prefix
((A * B) + (C / D))	((A B *) (C D /) +)	(+ (* A B) (/ C D))
((A * (B + C)) / D)	((A (B C +) *) D /)	(/ (* A (+ B C)) D)
(A * (B + (C / D)))	(A (B (C D /) +) *)	(* A (+ B (/ C D)))

Alıştırma

- Stack kullanarak sadece + ve – işlemcileri için INFIX ifadeyi POSTFIX veya PREFIX hale dönüştüren kodu yazınız…

Quiz – 1

- Kullanıcının gireceği 10 adet sayıyı stack(ler) kullanarak sıralı olarak saklayan programı yazınız…