Dynamiskt minne på heapen

Standardfunktioner `malloc`, `calloc`, `realloc` och `free` i `<stdlib.h>`

`size_t` är definierad i `<stddef.h>` som en heltalstyp med maxvärde >= 65535

Typen **`void`** `*` betecknar en pekare som kan typomvandlas till och från varje annan pekartyp.

```
void *malloc(size_t size);              // ger oinitierat utrymme
double *p = (double *) malloc(100 * sizeof(double));

void *calloc(size_t n, size_t size);  // ger nollställt utrymme
double *p = (double *) calloc(100, sizeof(double));

void *realloc(void *ptr, size_t size);  // ger större utrymme
*p = (double *) realloc(p, 200 * sizeof(double));
```

`malloc`, `calloc` och `realloc` ger `NULL` om det inte fanns tillräckligt med minne.

```
void free(void *ptr);          // återlämnar allokerat minne
free(p);
```

Vanliga fel vid användning av pekare:

1. utnyttjar en pekare som är oinitierad eller innehåller värdet `NULL`

2. utnyttjar en s.k. *kvardröjande pekare*, som pekar på ett minnesutrymme som tidigare varit allokerat men som blivit frisläppt

3. frisläpper inte minnesutrymme som inte längre behövs

```
// Ex fel av typ 1
char a[10];
char *p;
*a = 'X';  // OK! a[0] = 'X'
*p = 'X';  // Exekveringsfel!

// "a och p har ju samma typ"
```

```
// Ex fel av typ 2

char *read_line(int max)  // Felaktig funktion!!!
{
  char a[1000];    // Minnesutrymme på stacken
  char c;
  int  i = 0;
  while (i < max-1 && (c = getchar()) != '\n') {
    a[i] = c;
    i++;
  }
  a[i] = '\0';
  return a;         // Retur av kvardröjande pekare
}
```

```
// Ex fel av typ 3

char *read_line(int max)  // Olämplig funktion!!!
{
  char a[1000];    // Minnesutrymme på stacken
  char c;
  int  i = 0;
  while (i < max-1 && (c = getchar()) != '\n') {
    a[i] = c;
    i++;
  }
  a[i] = '\0';
  char *res = (char *) malloc((i+1)*sizeof (char));
  strcpy(res, a);
  return res;         // Retur av pekare till heap
}
```

```
// Hur gör man då?

void read_line(char *a, int max)
{
  char c;
  int  i = 0;
  while (i < max-1 && (c = getchar()) != '\n') {
    a[i] = c;
    i++;
  }
  a[i] = '\0';
}

// Anroparen ansvarar för minnesutrymmet

char b[30];
read_line(b, sizeof b);
```

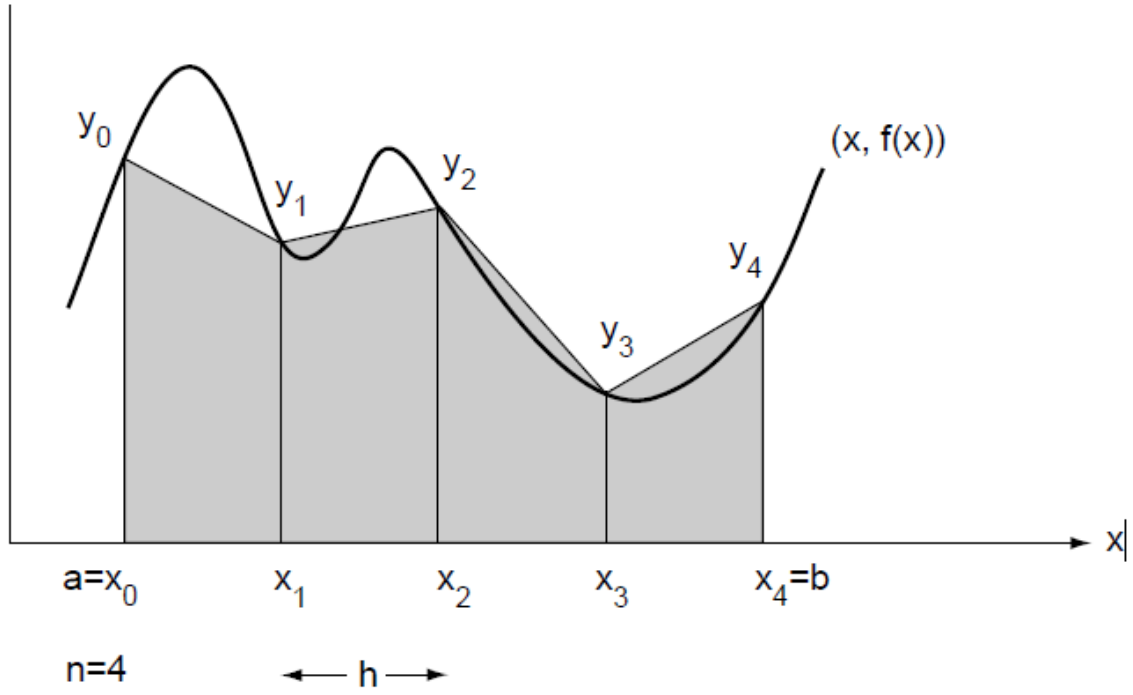
Pekare till funktioner

```
double (*funkptr)(double);

double trippel(double x) {
   return 3 * x;
}

funkptr = trippel;

double resultat;
resultat = funkptr(7.3);
```

Användningsområden:

- Parametrar till generella (matematiska) funktioner
- Callback-funktioner i grafiska användargränssnitt (jfr lyssnare i Java)
- Avbrottsrutiner vid maskinnära programmering

Ex. Beräkna integral av godtycklig funktion




```
#include <math.h>

resultat = trapets(0.0, 3.1415926535, sin, 4);
```

```
/* approximera integralen av funktionen f i intervallet [a, b]
   i n steg med trapetsmetoden */

double trapets(double a, double b, double (*f)(double), int n)
{
   int i;
   double sum, h = (b-a)/n;
   sum = f(a)/2 + f(b)/2;      /* se nedan */
   for (i=1; i<n; i++) {
      sum += f(a + i*h);
   }
   return h*sum;
}


sum = (*f)(a)/2 + (*f)(b)/2;    /* exakt samma som ovan */
```