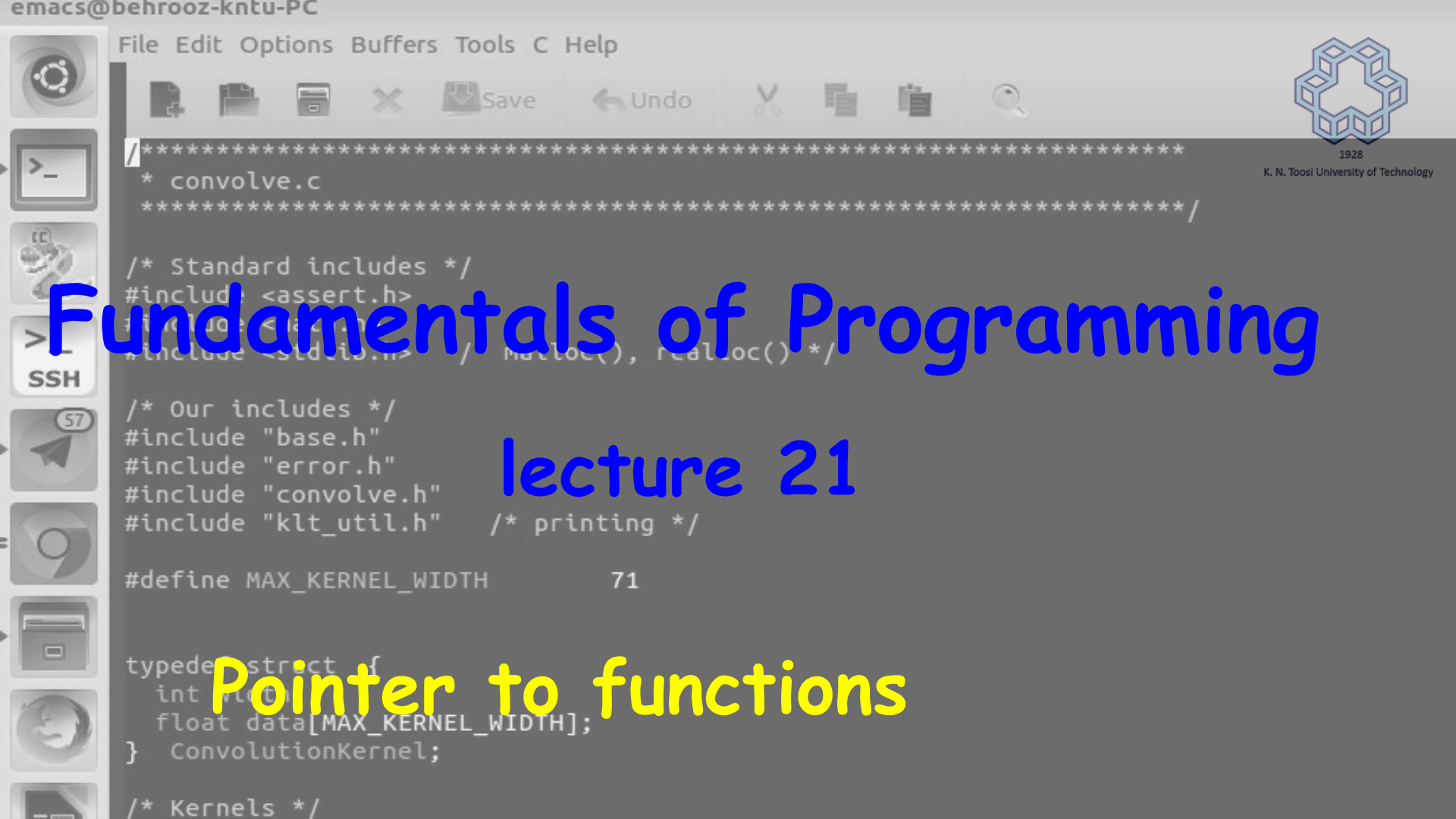
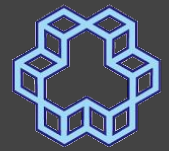

emacs@behrooz-kntu-PC
File Edit Options Buffers Tools C Help
Save
Undo
1928
K. N. Toosi University of Technology
/***
* convolve.c
***/
/* Standard includes */
/* Our includes */
#include "base.h"
#include "error.h"
#include "convolve.h"
#include "klt_util.h" /* printing */
#define MAX_KERNEL_WIDTH 71
float data[MAX_KERNEL_WIDTH];
/* Kernels */
Fundamentals of Programming
lecture 21
Pointer to functions

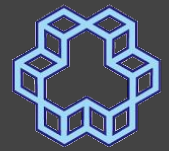
Example: compute derivatives




```c
#include <stdio.h>

double f(double x) { return x*x - 3*x + 2; }

double g(double x) { return 1/x; }

int main() {
  double delta = 1e-8; // = 1 * 10^-8

  double x = 2.0;

  double df_dx = ( f(x+delta) - f(x) ) / delta;

  double dg_dx = ( g(x+delta) - g(x) ) / delta;

  printf("df/dx at %.2f = %.2f\n", x, df_dx);
  printf("dg/dx at %.2f = %.2f\n", x, dg_dx);

  return 0;
}
```

arrayofpointers1.c

Example: compute derivatives

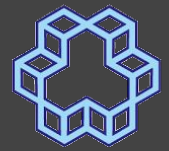



```c
#include <stdio.h>

double f(double x) { return x*x - 3*x + 2; }

double g(double x) { return 1/x; }

int main() {
  double delta = 1e-8; // = 1 * 10^-8

  double x = 2.0;
```

```
nasihatkon@kntu:code$ gcc pointertofunc1.c && ./a.out
df/dx at 2.00 = 1.00
dg/dx at 2.00 = -0.25
```

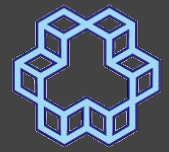
```c
  printf("df/dx at %.2f = %.2f\n", x, df_dx);
  printf("dg/dx at %.2f = %.2f\n", x, dg_dx);

  return 0;
}
```

The idea



Write a function that computes the derivative of any function!

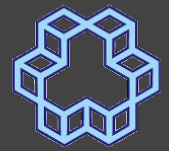
The idea



Write a function that computes the derivative of any function!

```
double x = 2.0;

double df_dx = derivative(f, x);

double dg_dx = derivative(g, x);
```

The idea

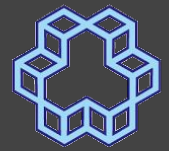


Write a function that computes the derivative of any function!

```
double x = 2.0;

double df_dx = derivative(f, x);

double dg_dx = derivative(g, x);
```

what is unusual about this?

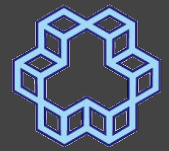
Pointer to function




```c
#include <stdio.h>

double f(double x) { return x*x-3*x+2;}

double g(double x) { return 1/x;}

int main() {

  double (*p)(double);

  p = &f;

  double x = 2.0;
  double y = (*p)(x);

  printf("x= %.2f, y= %.2f\n", x, y);

  return 0;
}
```

arrayofpointers3.c

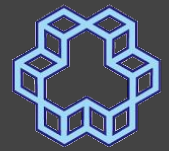
Pointer to function




```c
#include <stdio.h>

double f(double x) { return x*x-3*x+2;}

double g(double x) { return 1/x;}

int main() {

  double (*p)(double);

  p = &f;

  double x = 2.0;
  double y = (*p)(x);

  printf("x= %.2f, y= %.2f\n", x, y);

  retur
}
```

```
nasihatkon@kntu:code$ gcc pointertofunc3.c && ./a.out
x= 2.00, y= 0.00
```

Pointer to function



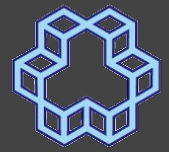

```c
#include <stdio.h>

double f(double x) { return x*x-3*x+2;}

double g(double x) { return 1/x;}

int main() {

  double (*p)(double);

  p = &f;

  double x = 2.0;
  double y = (*p)(x);

  printf("x= %.2f, y= %.2f\n", x, y);

  return 0;
}
```

arrayofpointers3.c

```c
#include <stdio.h>

double f(double x) { return x*x-3*x+2;}

double g(double x) { return 1/x;}

int main() {

  double (*p)(double);

  p = f;

  double x = 2.0;
  double y = p(x);

  printf("x= %.2f, y= %.2f\n", x, y);

  return 0;
}
```

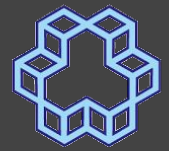
arrayofpointers4.c

Pointer to function




```
double (*p)(double);

p = f;

double x = 2.0;
double y = p(x);

p = g;
```

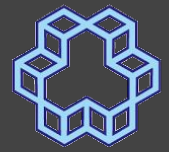
Passing function pointer as argument




```c
double derivate(double (*h)(double), double x) {
  double delta = 1e-8;
  return ( (*h)(x+delta) - (*h)(x) ) / delta;
}
```

arrayofpointers5.c

Passing function pointer as argument



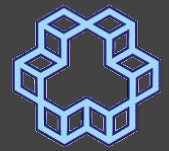

```c
double derivate(double (*h)(double), double x) {
  double delta = 1e-8;
  return ( (*h)(x+delta) - (*h)(x) ) / delta;
}
```

arrayofpointers5.c

```c
double derivate(double (*h)(double), double x) {
  double delta = 1e-8;

  return ( h(x+delta) - h(x) ) / delta;
}
```

arrayofpointers6.c

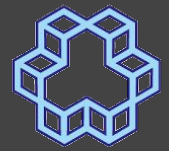
Passing function pointer as argument




```c
#include <stdio.h>

double f(double x) { return x*x-3*x+2;}

double g(double x) { return 1/x;}

double derivate(double (*h)(double), double x) {
  double delta = 1e-8;
  return ( h(x+delta) - h(x) ) / delta;
}

int main() {
  double x = 2.0;

  printf("df/dx at %.2f = %.2f\n", x, derivate(f,x));
  printf("dg/dx at %.2f = %.2f\n", x, derivate(g,x));

  return 0;
}
```

arrayofpointers6.c

Passing function pointer as argument



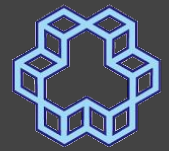

```c
#include <stdio.h>

double f(double x) { return x*x-3*x+2;}

double g(double x) { return 1/x;}

double derivate(double (*h)(double), double x) {
```

```
nasihatkon@kntu:code$ gcc pointertofunc6.c && ./a.out
df/dx at 2.00 = 1.00
dg/dx at 2.00 = -0.25
```

```c
  double x = 2.0;

  printf("df/dx at %.2f = %.2f\n", x, derivate(f,x));
  printf("dg/dx at %.2f = %.2f\n", x, derivate(g,x));

  return 0;
}
```

arrayofpointers6.c

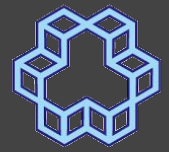
Passing function pointer as argument




```c
#include <stdio.h>
#include <math.h>

double f(double x) { return x*x-3*x+2;}
double g(double x) { return 1/x;}

double derivate(double (*h)(double), double x) {
  double delta = 1e-8;
  return ( h(x+delta) - h(x) ) / delta;
}

int main() {
  double x = 0.0;

  printf("d sin/dx at %.2f = %.2f\n", x, derivate(sin,x));

  return 0;
}
```

arrayofpointers7.c

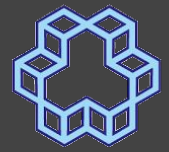
Passing function pointer as argument




```c
#include <stdio.h>
#include <math.h>

double f(double x) { return x*x-3*x+2;}
double g(double x) { return 1/x;}

double derivate(double (*h)(double), double x) {
  double delta = 1e-8;
  return ( h(x+delta) - h(x) ) / delta;
}

int main() {
  double x = 0.0;

  printf("d sin/dx at %.2f = %.2f\n", x, derivate(sin,x));
```

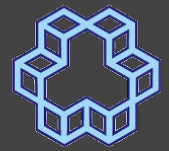
```
nasihatkon@kntu:code$ gcc pointertofunc7.c -lm && ./a.out
d sin/dx at 0.00 = 1.00
```

Remember "bubble sort"




```
void bubbleSort(int a[], int n) {
  int m;

  for (m = n-1; m > 0; m--) {
    for (int i = 0; i < m; i++) {

      if (a[i] > a[i+1]) {
        swap(&a[i], &a[i+1]);
      }


    }
  }
}
```

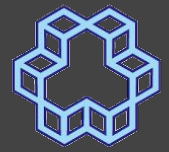
Remember "bubble sort"




```
void bubbleSort(int a[], int n) {
  int m;

  for (m = n-1; m > 0; m--) {
    for (int i = 0; i < m; i++) {

      if (a[i] > a[i+1]) {
        swap(&a[i], &a[i+1]);
      }


    }
  }
}
```

Sort both ascendingly and descendingly?

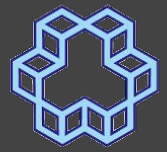
control sort order




```
void bubbleSort(int a[], int n) {
  int m;

  for (m = n-1; m > 0; m--) {
    for (int i = 0; i < m; i++) {

      if (a[i] > a[i+1]) {

        swap(&a[i], &a[i+1]);
      }


    }
  }
}
```

```
void bubbleSort(int a[], int n) {
  int m;

  for (m = n-1; m > 0; m--) {
    for (int i = 0; i < m; i++) {

      if ( must_swap(a[i], a[i+1]) ) {

        swap(&a[i], &a[i+1]);
      }


    }
  }
}
```

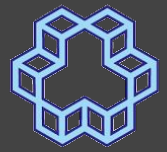
control sort order




```c
void bubbleSort(int a[], int n, int (*must_swap)(int, int) ) {
  int m;

  for (m = n-1; m > 0; m--) {
    for (int i = 0; i < m; i++) {

      if ( must_swap(a[i], a[i+1])  ) {

        swap(&a[i], &a[i+1]);
      }


    }
  }
}
```

pointertofuncsort.c

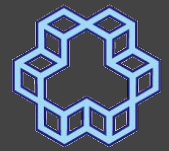
sort with pointer to function




```c
void printArray(int a[], int n);
void bubbleSort(int a[], int n, int (*must_swap)(int, int));
void swap(int *p, int *q);

int must_swap_ascending(int a, int b) {
  return a > b;
}

int must_swap_descending(int a, int b) {
  return a < b;
}
```

pointertofuncsort.c

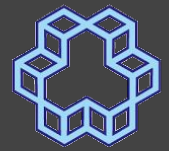
control sort order




```c
int main() {
  int a[] = {25,19,14,18,27,6,32,18,20,1,21};
  int n = sizeof(a) / sizeof(a[0]);

  printArray(a,n);

  bubbleSort(a, n, must_swap_ascending);

  printArray(a,n);

  bubbleSort(a, n, must_swap_descending);

  printArray(a,n);


  return 0;
}
```

pointertofuncsort.c

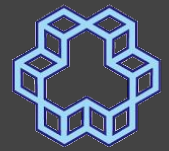
control sort order




```c
int main() {
  int a[] = {25,19,14,18,27,6,32,18,20,1,21};
  int n = sizeof(a) / sizeof(a[0]);

  printArray(a,n);

  bubbleSort(a, n, must_swap_ascending);

  printArray(a,n);

  bubbleSort(a, n, must_swap_descending);

  printArray(a,n);
```

```
nasihatkon@kntu:code$ gcc pointertofuncsort2.c && ./a.out
25, 19, 14, 18, 27, 6, 32, 18, 20, 1, 21,
1, 6, 14, 18, 18, 19, 20, 21, 25, 27, 32,
32, 27, 25, 21, 20, 19, 18, 18, 14, 6, 1,
```

array of pointer to functions

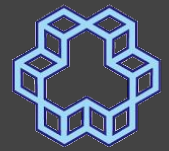


pointertofuncarray.c

```c
void print(int x) {
  printf("x= %d\n", x);
}

void print_square(int x) {
  printf("x^2= %d\n", x*x);
}

void print_cube(int x) {
  printf("x^3= %d\n", x*x*x);
}
```

```c
void (*funcArray[3])(int) = {print, print_square, print_cube};
```

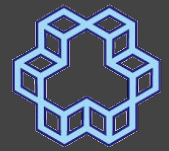
array of pointer to functions



pointertofuncarray.c

```c
int main() {
  int x = 4;

  void (*funcArray[3])(int) = {print, print_square, print_cube};

  for (int i = 0; i < 3; i++) {
    funcArray[i](x);
  }

  return 0;
}
```

array of pointer to functions



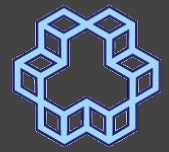
pointertofuncarray.c

```c
int main() {
  int x = 4;

  void (*funcArray[3])(int) = {print, print_square, print_cube};

  for (int i = 0; i < 3; i++) {
    funcArray[i](x);
  }
```

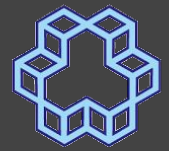
```
nasihatkon@kntu:code$ gcc pointertofuncarray.c && ./a.out
x= 4
x^2= 16
x^3= 64
```

How to interpret?

```
void (*f[3])(int);
```



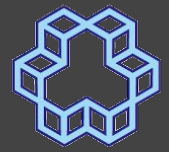
How to interpret?




```
void (*f[3])(int);
```

```
void f(int);
```

How to interpret?

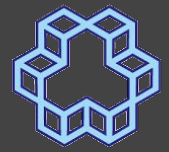



```
void (*f[3])(int);
```

```
void f(int);
```

function of int returning nothing

How to interpret?



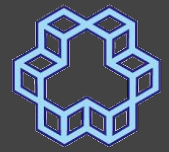

```
void (*f[3])(int);
```

```
void f(int);
```

function of int returning nothing

```
void (*f)(int);
```

How to interpret?

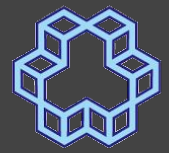



```
void (*f[3])(int);
```

```
void f(int);
```
function of int returning nothing

```
void (*f)(int);
```
pointer to func. of int returning nothing

How to interpret?



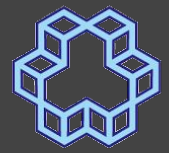

```
void (*f[3])(int);
```

```
void f(int);
```
function of int returning nothing

```
void (*f)(int);
```
pointer to func. of int returning nothing

```
void (*f[3])(int);
```

How to interpret?




```
void (*f[3])(int);
```

```
void f(int);
```
function of int returning nothing

```
void (*f)(int);
```
pointer to func. of int returning nothing

```
void (*f[3])(int);
```
array of size 3 to pointer of func. of int returning nothing

.

precedence table

```
int (*f[3])(int);

int *f[3](int);
```

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right