Int x; -> 1 element

Int x[5]; -> 5 element

All 5 elements of type int

Int x[5][2]; -> 5x2 = 10 elements

All 5 elements of type int x[3] and each element of those 5 elements are pf type int

Int x[5][2][3]; -> 5x2x3 = 30 elements

All 5 elements are of type int [2][3]; each of those elements are of type int [3] and each of those elements is type int.

Example x[5][3] elements ordering.

#include<stdio.h>

int main(){

int x[5][3];

// in all 15 elements of type int

// x is an array of 5 elements of type int x[3]

// each of those 5 elelemnts is of type int

int \*j;

int y, total;

x[0][0]=1;

x[0][1]=2;

x[0][2]=3;

x[1][0]=10;

x[1][1]=20;

x[1][2]=30;

x[2][0]=100;

x[2][1]=200;

x[2][2]=300;

x[3][0]=1000;

x[3][1]=2000;

x[3][2]=3000;

x[4][0]=10000;

x[4][1]=20000;

x[4][2]=30000;

j = (int \*)x; // will be converted to j= (int \*)&x[0];

for(total = 0, y=0; y<=14; y++){

total = total+ \*j;

printf("added %d\n", \*j);

j++;

}

printf("Total is %d\n", total);

return 0;

}

Output:

#include<stdio.h>

int main(){

int x[5][3];

// in all 15 elements of type int

// x is an array of 5 elements of type int x[3]

// each of those 5 elelemnts is of type int

int \*j;

int y, total;

x[0][0]=1;

x[0][1]=2;

x[0][2]=3;

x[1][0]=10;

x[1][1]=20;

x[1][2]=30;

x[2][0]=100;

x[2][1]=200;

x[2][2]=300;

x[3][0]=1000;

x[3][1]=2000;

x[3][2]=3000;

x[4][0]=10000;

x[4][1]=20000;

x[4][2]=30000;

j = (int \*)x; // will be converted to j= (int \*)&x[0];

for(total = 0, y=0; y<=14; y++){

total = total+ \*j;

printf("added %d\n", \*j);

j++;

}

printf("Total is %d\n", total);

return 0;

}

/\*

added 1

added 2

added 3

added 10

added 20

added 30

added 100

added 200

added 300

added 1000

added 2000

added 3000

added 10000

added 20000

added 30000

Total is 66666

\*/

We will get same output with

for(total = 0, y=0; y<=14; y++){

total = total+ \*(j+y);

printf("added %d\n", \*(j+y));

// j++;

}

&x = 700

J = &x;

\* ( j + 1) = 704

\* ( j + 2) = 708

\* ( j + 3) = 712

Write seprate and common function to sum

#include<stdio.h>

int calculate\_sum(int \*j, int rows, int columns){

int total,y;

int total\_elements = rows\*columns;

for ( total = 0, y = 0; y<total\_elements; y++)

total = total + \*(j+y);

return total;

}

int main(){

int x[5][3], total;

x[0][0]=1;

x[0][1]=2;

x[0][2]=3;

x[1][0]=10;

x[1][1]=20;

x[1][2]=30;

x[2][0]=100;

x[2][1]=200;

x[2][2]=300;

x[3][0]=1000;

x[3][1]=2000;

x[3][2]=3000;

x[4][0]=10000;

x[4][1]=20000;

x[4][2]=30000;

total = calculate\_sum((int \*)x, 5,3);

printf("Total is %d\n", total);

return 0;

}

/\*

added 1

added 2

added 3

added 10

added 20

added 30

added 100

added 200

added 300

added 1000

added 2000

added 3000

added 10000

added 20000

added 30000

Total is 66666

\*/

Also applicable for 1d 2d array

#include<stdio.h>

int calculate\_sum(int \*j, int rows, int columns){

int total,y;

int total\_elements = rows\*columns;

for ( total = 0, y = 0; y<total\_elements; y++)

total = total + \*(j+y);

return total;

}

int main(){

int x[5], total,y;

for(y=0; y<5; y++){

printf("Enter a number: ");

scanf("%d", &x[y]);

}

total = calculate\_sum((int \*)x, 1,5);

printf("Total is %d\n", total);

return 0;

}

Points can be used as an array

// pointer can be used as an array

#include<stdio.h>

int calculate\_sum(int \*j, int rows, int columns){

int total,y;

int total\_elements = rows\*columns;

for ( total = 0, y = 0; y<total\_elements; y++)

// total = total + \*(j+y);

total = total + j[y];

return total;

}

int main(){

int x[5], total,y;

for(y=0; y<5; y++){

printf("Enter a number: ");

scanf("%d", &x[y]);

}

total = calculate\_sum((int \*)x, 1,5);

printf("Total is %d\n", total);

return 0;

}

MOST IMPORTANT POINTS

When you use pointers as an array, we should always provide the base address only.

Otherwise whole arithmetic calculation will be wrong.

Lets take an example to understand above statement.

#include<stdio.h>

int main(){

int x[5];

int y;

int \*p;

p = x;

p[0] = 100;

p++; //increaseing the value of P here

p[0] = 200;

p[1] = 300;

p[2] = 400;

p[3] = 500;

for(y=0; y<5; y++){

printf("%d\n", x[y]);

}

return 0;

}

/\*

Output:

100

200

300

400

500

\*/

Here we can see p[0] initllay point to 1st element of x

But when increase the value of p

Then again we write p[0] its pointing to 1st element of x. Check and above example

Going out of bounds

Code will compile

BUT

Application may or may not crash (UNDEFINED BEHAVIOUR)

#include<stdio.h>

int main(){

int x[5];

x[100000] = 200;

printf("%d\n", x[100000]);

return 0;

}

/\*

zsh: segmentation fault eg

\*/