**Passing and returning values using pointers,**

Pointers give greatly possibilities to 'C' functions which we are limited to return one value. With pointer parameters, our functions now can process actual data rather than a copy of data.

**Functions Pointers**

void swap (int \*, int \*);

int main()

{

int a = 10;

int b = 20;

printf("a is %d, b is %d\n", a, b);

swap(&a, &b);

printf("a is %d, b is %d\n", a, b);

return 0;

}

void swap (int \*x, int \*y)

{

int t;

t = \*x;

\*x = \*y;

\*y = t;}

}

**Functions with Array Parameters**

int add\_array (int \*, int );

int main()

{

int mark[5] = {80, 90, 75, 80, 90};

printf("Sum of Mark is %d\n", sum(mark, 5));

return 0;

}

int sum (int m[], int size)

{

int total = 0;

for (int i = 0; i < size; i++) {

total += m[i]; /\* it is equivalent to total +=\*p ;p++; \*/

}

return (total);

}

**Functions that Return an Array**

#include<stdio.h>

int\* getmark();

int main()

{

int \*m;

m=getmark();

for (int i = 0; i < 5; i++)

printf("%d\n", m[i]);

return 0;

}

int\* getmark()

{

static int mark[5] = {80, 90, 75, 80, 90}; //Why static

return (mark);

}

**Function returning pointer**

#include <stdio.h>

int\* fun()

{

static int A = 10; //why static

return (&A);

}

int main()

{

int\* p;

p = fun();

printf("%d\n", p);

printf("%d\n", \*p);

return 0;

}

**Functions returning Pointer variables**

#include <stdio.h>

int\* larger(int\*, int\*);

void main()

{

int a = 10;

int b = 20;

int \*p;

p = larger(&a, &b);

printf("%d is larger",\*p);

}

int\* larger(int \*x, int \*y)

{

if(\*x > \*y)

return x;

else

return y;

}

**Pointer to a function**

C allows operations with pointers to functions. Every function code along with its variable is allocated some space in the memory. Every function has an address.

Function pointers are pointer variables that point to the address of a function. Function pointers can be declared, assignd values, and used to access the functions they point to.

The general statement of its declaration is

return\_type (\*function\_name) (arguments)

.

int (\*func) (int a float b)

Steps

1. Declare the function pointer.
2. Assign a function to the function pointer

**Calling a function through function pointer**

Example 1:

#include <stdio.h>

void fun(int a)

{

printf("Value of a is %d\n", a);

}

int main()

{

void (\*fun\_ptr)(int);

fun\_ptr=&fun;

(\*fun\_ptr)(10);

return 0;

}

**1)** Unlike normal pointers, a function pointer points to code, not data. Typically a function pointer stores the start of executable code.

**2)**Unlike normal pointers, we do not allocate de-allocate memory using function pointers.

**3)** A function’s name can also be used to get functions’ address. For example, in the below program, we have removed address operator ‘&’ in assignment. We have also changed function call by removing \*, the program still works.

Example 2:

#include <stdio.h>

void fun(int a)

{

printf("Value of a is %d\n", a);

}

int main()

{

void (\*fun\_ptr)(int);

fun\_ptr=fun;

(fun\_ptr)(10);

return 0;

}

Example 3:

Ilustrates how the function pointer can be changed to point different function during the program execution

#include <stdio.h>

int sum(int n1, int n2);

int sub(int n1, int n2);

int mult(int n1, int n2);

int div(int n1, int n2);

int main()

{

int x,y,ch,r;

int (\*ope)(int, int);

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

ope = sum;

r = ope(x,y);

printf("%d\n", r);

ope = sub;

r = ope(x,y);

printf("%d\n", r);

ope = mult;

r = ope(x,y);

printf("%d\n", r);

ope = div;

r = ope(x,y);

printf("%d\n", r);

return 0;

}

int sum(int x,int y)

{

return(x + y);

}

int sub(int x,int y)

{

return(x - y);

}

int mult(int x,int y)

{

return(x \* y);

}

int div(int x,int y)

{if (y != 0)

return (x / y);

else return 0;

}

**4)** Like normal pointers, we can have an **array of function pointers**.  Function pointer can be used in place of switch case.

Example 4:

#include <stdio.h>

int sum(int n1, int n2);

int sub(int n1, int n2);

int mult(int n1, int n2);

int div(int n1, int n2);

int main()

{

int x,y,ch,r;

int (\*ope[4])(int, int);

ope[0] = sum;

ope[1] = sub;

ope[2] = mult;

ope[3] = div;

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

printf("Enter 0 to sum, 1 to subtract, 2 to multiply, or 3 to divide: ");

scanf("%d", &ch);

r = ope[ch](x, y);

printf("%d", r);

return 0;

}

int sum(int x,int y)

{

return(x + y);

}

int sub(int x,int y)

{

return(x - y);

}

int mult(int x,int y)

{

return(x \* y);

}

int div(int x,int y)

{if (y != 0)

return (x / y);

else return 0;

}

**Passing a Function pointer as an argument to a function**

Example 5:

#include <stdio.h>

int sum(int n1, int n2);

int sub(int n1, int n2);

int (ope)(int (\*op\_funcp)(int,int),int,int);

int main()

{

int r,x,y;

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

r = ope(sum,x,y);

printf("%d\n", r);

r = ope(sub,x,y);

printf("%d\n", r);

return 0;

}

int sum(int x,int y)

{

return(x + y);

}

int sub(int x,int y)

{

return(x - y);

}

int (ope)(int (\*op\_funcp)(int,int),int a,int b)

{

int result;

result=(op\_funcp)(a,b);

return result;

}