The syntax for declaring a function pointer might seem messy at first, but in most cases it's really quite straight-forward once you **understand** what's going on. Let's look at a simple example:

void (\*foo)(int);

In this example, foo is a pointer to a function taking one argument, an integer, and that returns void. It's as if you're declaring a function called "\*foo", which takes an int and returns void; now, if \*foo is a function, then foo must be a pointer to a function. (Similarly, a declaration like int \*x can be read as \*x is an int, so x must be a pointer to an int.)   
  
The key to writing the declaration for a function pointer is that you're just writing out the declaration of a function but with (\*func\_name) where you'd normally just put func\_name.

**Reading Function Pointer Declarations**

Sometimes people get confused when more stars are thrown in:

void\*(\*foo)(int \*);

Here, the key is to read inside-out; notice that the innermost element of the expression is \*foo, and that otherwise it looks like a normal function declaration. \*foo should refer to a function that returns a void \* and takes an int \*. Consequently, foo is a pointer to just such a function.

**Initializing Function Pointers**

To initialize a function pointer, you must give it the address of a function in your program. The syntax is like any other variable:

#include <stdio.h>

voidmy\_int\_func(int x)

{

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

}

int main()

{

void (\*foo)(int);

/\* the ampersand is actually optional \*/

foo = &my\_int\_func;

return 0;

}

**Using a Function Pointer**

To call the function pointed to by a function pointer, you treat the function pointer as though it were the name of the function you wish to call. The act of calling it performs the dereference; there's no need to do it yourself:

#include <stdio.h>

voidmy\_int\_func(int x)

{

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

}

int main()

{

void (\*foo)(int);

foo = &my\_int\_func;

/\* call my\_int\_func (note that you do not need to write (\*foo)(2) ) \*/

foo( 2 );

/\* but if you want to, you may \*/

(\*foo)( 2 );

return 0;

}

Note that function pointer syntax is flexible; it can either look like most other uses of pointers, with & and \*, or you may omit that part of syntax. This is similar to how arrays are treated, where a bare array decays to a pointer, but you may also prefix the array with & to request its address.

#include<stdio.h>

intfunc (int a, int b)

{

printf("\n a = %d\n",a);

printf("\n b = %d\n",b);

return 0;

}

int main(void)

{

int(\*fptr)(int,int); // Function pointer

fptr = func; // Assign address to function pointer

func(2,3);

fptr(2,3); //(\*fptr)(2,3)

return 0;

}

#include<stdio.h>

#include<conio.h>

int a(inta,int b)

{

returna+b;

}

int s(inta,int b)

{

return a-b;

}

voidmethfun(intx,inty,int (\*sum)(int,int))

{

intans=(\*sum)(x,y);//calling function (if a it will call a function)

//in tans=sum(x,y);

printf("%d",ans);

}

void main()

{

intx,y,ans;

char c;

printf("enter two no");

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

printf("enter choice 'a' -'s'");

c=getche();

if(c=='a')

methfun(x,y,a); calling function

else if(c=='s')

methfun(x,y,s);

getche();

}

#include<stdio.h>

// Note our user-defined comparison is the third parameter

Void SelectionSort(int \*arr, int nSize, int (\*pComparison)(int,int))

{

int t;

for (int i= 0; i <nSize-1; i++)

{

// Search through every element starting at nStartIndex+1

for (int j = i+ 1; j <nSize; j++)

{

// Note that we are using the user-defined comparison here

if (pComparison(arr[i], arr[j])) // COMPARISON DONE HERE

{

t=arr[i];

arr[i]=arr[j];

arr[j]=t;

}

}

}

}

// Here is a comparison function that sorts in ascending order

// (Note: it's exactly the same as the previous Ascending() function)

int Ascending(int nX, int nY)

{

Return nX>nY;

}

// Here is a comparison function that sorts in descending order

int Descending(intnX, intnY)

{

Return nX<nY;

}

// This function prints out the values in the array

voidPrintArray(int \*pArray, intnSize)

{

for (int iii=0; iii <nSize; iii++)

printf("\n%d", pArray[iii] );

}

int main()

{

Int arr[9] = { 3, 7, 9, 5, 6, 1, 8, 2, 4 };

// Sort the array in descending order using the Descending() function

//SelectionSort(arr, 9, Descending);

//PrintArray(arr, 9);

// Sort the array in ascending order using the Ascending() function

SelectionSort(arr, 9, Ascending);

PrintArray(arr, 9);

return 0;

}