Types of Functions in C Programming

Tom Penick tomzap@eden.com www.teicontrols.com 1/28/98

The following examples illustrate various methods of passing values to functions. Except for the function "strcopy()", these are not working functions (code has been omitted).

Contents

A FUNCTION WHICH PASSES NO VALUE AND RETURNS NO VALUE

A function may be declared (function prototype) globally or within the calling function:

FUNCTION PROTOTYPE
```
void PrintHead(void);
```
FUNCTION CALL
```
PrintHead();
```
FUNCTION HEADER
```
void PrintHead(void)
{
```
RETURN STATEMENT
```
return;
}
```

A FUNCTION WHICH PASSES TWO FLOATS AND RETURNS A FLOAT

A function can *return* at most one value:

FUNCTION PROTOTYPE
```
float find_max(float, float);
```
FUNCTION CALL
```
maxmum = find_max(firstnum,secnum);
```

The variables used in the function need not and should not have the same names as those passed to the function:

FUNCTION HEADER
```
float find_max(float num1, float num2)
{
```
DECLARE A VARIABLE
```
float Result;
```
RETURN STATEMENT
```
return(Result);
}
```

A FUNCTION WHICH PASSES AN INTEGER ARRAY AND RETURNS AN INTEGER

An alternate method would be to pass by reference using a pointer. In this example the last argument is an integer telling the function how many elements are in the array:

FUNCTION PROTOTYPE
```
int find_max(int vals[], int);
```
FUNCTION CALL
```
biggun = find_max(nums,5);
```

The variables used in the function need not and should not have the same names as those passed to the function:

FUNCTION HEADER
```
int find_max(int nums[], int HowMany)
{
```
DECLARE A VARIABLE
```
int Result;
```
RETURN STATEMENT
```
return(Result);
}
```

A FUNCTION WHICH PASSES VARIABLES BY REFERENCE USING ADDRESSES

FUNCTION PROTOTYPE
```
void sortnum(double*, double*);
```
FUNCTION CALL
```
sortnum(&FirstNum, &SecNum);
```
FUNCTION HEADER
```
void sortnum(double *Num1, double *Num2)
{
```
RETURN STATEMENT
```
return;
}
```

A FUNCTION WHICH PASSES A STRING BY REFERENCE

There is no way that I can find of returning a string from a function. However, if the address of the string is passed, then the function can operate on the string. This example is a working function which takes the string referenced by the second argument, removes the carriage return from the end of it and "returns" it by assignment to the first argument. (This is used for a string which has been retrieved from a text file using the `fgets()` function):

FUNCTION PROTOTYPE

```
void strcopy(char [], char []);
```

The calling function must have declared two appropriate character arrays.

```
char Name1[25];
char Name2{25];
```

FUNCTION CALL

```
strcopy(Name2,Name1);
```

FUNCTION HEADER

```
void strcopy(char Str2[], char Str1[])
{
```

DECLARE A VARIABLE

```
int Cnt = 0;
while (Str1[Cnt] != '\n')
{
     Str2[Cnt] = Str1[Cnt];
     ++Cnt;
}
```

Nothing is returned, but "Str2" is the new version of the original "Name1" and is available in the calling function as "Name2".

RETURN STATEMENT

```
return;
}
```

A FUNCTION WHICH PASSES A STRUCTURE BY NAME

Here "class_list" is a structure type declared globally:

STRUCTURE DECLARATION

```
struct class_list
{
     char Name[31];
     long ID_Num;
     char Class[9]
};
```

The function prototype may be declared globally or within the calling function. Here "class_list" is the type of structure from the structure prototype (declared globally), not the specific structure itself:

FUNCTION PROTOTYPE

```
void PrintReport(struct class_list);
```

A single structure of type "class_list" is created in the calling function (if not globally) and named "load":

STRUCTURE IS CREATED

```
struct class_list load;
```

The structure "load" is passed to the function:

FUNCTION CALL

```
PrintReport(load);
```

The structure prototype name is again used in the function header:

FUNCTION HEADER

```
void PrintReport(struct class_list N)
{
```

REFERENCES TO ELEMENTS

```
N.Name
N.ID_Num
N.Class
```

RETURN STATEMENT

```
return;
}
```

A FUNCTION WHICH PASSES A STRUCTURE BY REFERENCE USING A POINTER

Here "class_list" is a structure type declared globally as before:

STRUCTURE DECLARATION

```
struct class_list
{
     char Name[31];
     long ID_Num;
     char Class[9]
};
```

The function prototype may be declared globally or within the calling function. Here "class_list" is the type of structure from the structure prototype (declared globally), not the specific structure itself. The * indicates that a pointer to the structure will be passed:

FUNCTION PROTOTYPE

```
void PrintReport(struct class_list *);
```

A single structure of type "class_list" is created in the calling function (if not globally) and named "load":

STRUCTURE IS CREATED

```
struct class_list load;
```

The structure is assigned to a pointer.

A POINTER IS DECLARED

```
struct class_list *Ptr;
```

A POINTER IS ASSIGNED

```
Ptr = &load;
```

The pointer to the structure is passed to the function.

FUNCTION CALL

```
PrintReport(Ptr);
```

A corresponding pointer "P" is declared in the function header:

FUNCTION HEADER

```
void PrintReport(struct class_list *P)
{
```

REFERENCES TO ELEMENTS

```
P->Name
P->ID_Num
P->Class
```

RETURN STATEMENT

```
return;
}
```

A FUNCTION WHICH PASSES A STRUCTURE ARRAY

Here "c_list" is a structure type declared globally as before:

STRUCTURE DECLARATION

```
struct c_list
{
     char Name[31];
     long ID_Num;
     char Class[9]
};
```

The function prototype may be declared globally or within the calling function. Here "c_list" is the type of structure from the structure prototype (declared globally), not the specific structure itself. The * indicates that a pointer to the structure will be passed:

FUNCTION PROTOTYPE

```
void PrintReport(struct c_list *);
```

A pointer to a structure of type "c_list" is created.

A POINTER IS DECLARED

```
struct c_list *Ptr;
```

A structure array of type "c_list" is created in the calling function and assigned to pointer "Ptr" and memory is allocated. "Elements" is the number of elements in the array:

STRUCTURE ARRAY IS CREATED

```
Ptr = (struct c_list *) malloc(Elements * sizeof(struct c_list));
```

The pointer to the structure is passed to the function.

FUNCTION CALL

```
PrintReport(Ptr);
```

A corresponding pointer "P" is declared in the function header:

FUNCTION HEADER

```
void PrintReport(struct c_list *P)
{
```

REFERENCES TO ELEMENTS

```
P[i].Name
P[i].ID_Num
P[i].Class
```

RETURN STATEMENT

```
return;
}
```

A FUNCTION WHICH PASSES A FILE NAME

A file pointer is declared in the calling function:

POINTER DECLARATION
FILE NAME ASSIGNMENT

```
FILE *Data;
Data = fopen("class.dat", "r+");
```

The argument is a pointer to a file:

FUNCTION PROTOTYPE
FUNCTION CALL

```
void ReadFile(FILE *)
ReadFile(Data);
```

A new file pointer is declared in the function header:

FUNCTION HEADER
RETURN STATEMENT

```
void ReadFile(FILE *F)
{
return;
}
```