C programming

Function

Section 1

What is function

What is function

A function is a self-contained program segment that carries out a specific, well-defined task

Functions are generally used as abbreviations for a series of instructions that are to be executed more than once

Functions are easy to write and understand

Debugging the program becomes easier as the structure of the program is more apparent, due to its modular form

Programs containing functions are also easier to maintain, because modifications, if required, are confined to certain functions within the program

Syntax

- The general syntax of a function in C is :

```
type_specifier function_name (arguments)
{
        body of the function
}
```

The type_specifier specifies the data type of the value, which the function will return.

A valid function name is to be assigned to identify the function

Arguments appearing in parentheses are also termed as formal parameters.

Section 2

Function parameter

Argument

The program calculates the square of numbers from 1 to 10

```
#include <stdio.h>
main()
{
    int i;
    for (i =1; i <=10; i++)
    printf ("\nSquare of %d is %d ", i,squarer (i));
}

squarer (int x)
/* int x; */
{
    int j;
    j = x * x;
    return (j);
}
```

The function works on data using arguments.

The data is passed from the main() to the **squarer()**

Returning from function

```
squarer (int x)
/* int x; */
{
    int j;
    j = x * x;
    return (j);
}
```

It transfers the control from the function back to the calling program immediately.

Whatever is inside the parentheses following the return, statement is returned as a value to the calling program.

Invoking a function

A semicolon is used at the end of the statement when a function is called, but not after the function definition

Only one value can be returned by a function

The function that calls another function is known as the calling function/routine

Parentheses are compulsory after the function name, irrespective of whether the function has arguments or not

The program can have more than one function

The function being called is known as the called function/routine

Function declaration

Function protoype

```
int foo(int arg1, char arg2);
```

Extern keyword

```
extern int foo(int arg1, char arg2);
```

Function prototype

- Use in <file>.c or <file>.h

int foo(int arg1, char arg2);

extern int foo(int arg1, char arg2);

Similar to function definition but without content

Very important because it inform compiler what the identifying word means, and how the identified thing should be used

a "declaration" refers only to a pure declaration (types only, no value or body), while a "definition" refers to a declaration that includes a value or body

Example

- Trainer use table to create an example how to use prototype and extern keyword
- Open question: Trainer need to guide Trainee how to think about different on usage of prototype and extern

Variable

Local Variables	• Declared inside a function • Created upon entry into a block and destroyed upon exit from the block
Formal Parameters	• Declared in the definition of function as parameters • Act like any local variable inside a function
Global Variables	• Declared outside all functions • Holds value throughout the execution of the program

Function scope

Global scope meaning function is visible or using in multiple translate unit.

Internal scope means function is only using in single translate unit

Passing argument by value

Foo_func(val1, val2)

Foo_func(10, 12)

10

12

Create a copy of val1 and val2

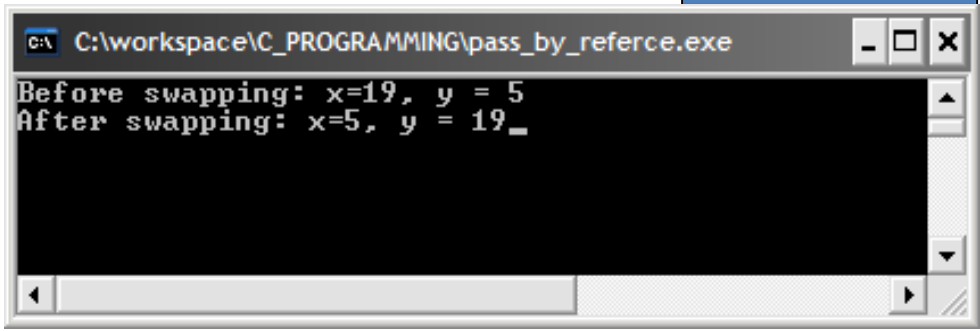
```
void Foo_func (int val1, int val2)
{
    val1 += val2;
}
```

Copy of val1 and val2 will be stored in private space.
This could be stack or temporary core register

Example swap function

```c
/* Pass-by-Reference example */
#include <stdio.h>
int swap (int *a, int *b);
int main ()
{
  int x = 19, y = 5;
  printf("Before swapping: x=%d, y = %d\n",x,y);
  swap(&x, &y);
  printf("After swapping: x=%d, y = %d",x,y);
  return 0;
}
int swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

Output



Passing argument by reference

Foo_func(ptr2var1, ptr2var2)

Foo_func(&a, &b)

&a

&b

Copy address of variable a and variable b

```
void Foo_func (int * val1, int * val2)
{
    *val1 += *val2;
}
```

Address of a and b are passed into function private space. This address will be change using pointer method inside function

Example usage

- Usage of passing by reference
  - ✓ Greater time and space efficiency – example (function with many input params)
  - ✓ Function can change value of argument, change reflect in calling function
  - ✓ Function with multiple output

Section 3

Function keyword

Introduction to keyword

- Static
  - ✓ A static function in C is a function that has a scope that is limited to its object file. This means that the static function is only visible in its object file
  - ✓ Syntax

```
static void staticFunc(void)
{
    printf("Inside the static function staticFunc() ");
}
```

Introduction to keyword

- Static
  - ✓ A static function in C is a function that has a scope that is limited to its object file. This means that the static function is only visible in its object file
  - ✓ Static functions are functions that are only visible to other functions in the same file (more precisely the same translation unit).

Introduction to keyword

THINK?

Introduction to keyword

- Inline
  - ✓ Inline Function are those function whose definitions are small and be substituted at the place where its function call is happened. Function substitution is totally compiler choice.
  - ✓ Syntax:

```
inline int foo()
    {
        return 2;
    }
```

Introduction to keyword

Inline

- Speed: Inline keyword is encourage compiler to build a function to code where it used (inside other function).
- Size: will be duplicated per each call

Normal function

- Speed: Function is place in different segment of memory, required jump to move from current place of call to place of function
- Size: Only one place, unique and compact.

Introduction to keyword

```c
#include <stdio.h>

// Inline function in C
inline int foo()
{
    return 2;
}

// Driver code
int main()
{

    int ret;

    // inline function call
    ret = foo();

    printf("Output is: %d\n", ret);
    return 0;
}

```

- Can we extern inline function and use it in other file?

Section 4

Function-like macro

Function-like macro

- Function-like macros can take *arguments*, just like true functions. To define a macro that uses arguments, you insert *parameters* between the pair of parentheses in the macro definition that make the macro function-like. The parameters must be valid C identifiers, separated by commas and optionally whitespace
- syntax:
  - ✓ #define Macro_Function_name(param0, param1,...) (expression)

Function-like macro

Example:

- #define min(X, Y) ((X) < (Y) ? (X) : (Y))

```
x = min(a, b);
```

```
x = ((a) < (b) ? (a) : (b));
```

```
y = min(a + 28, *p);
```

```
y = ((a + 28) < (*p) ? (a + 28) : (*p));
```

Function-like macro

Open question:

- What happen if parameter is not wrapped inside parentheses?
- Type checking
- Can Function-like macro can be pointed by pointer
- Compare against inline function and static function

Section 5

Recursion

Section 6

Variable argument list

Lesson Summary

- *Student (Trainee) try to list again what they have learn through section.*
- *Trainer help to summarize what they need to remember and continue expanding their knowledge*
- *Trainer can help to answer some open question.*

Thank you