C Programing

Trainer : Kiran Jaybhave

Loops - Iterations:

- Helps to give execution control repeatedly to a specific block
- 1. On Entry Check Loop
  - 1. While
  - 2. For
- 2. On Exit Check Loop
  - 1. do…while

- **Please note in all loops available in C language execution control will enter inside loop block only when given entry/exit given expression will result true.**
- While writing loop program should focus on
  - Initial state
  - Expression Check
  - Modification Statement – is a statement which helps to result expression to false state
  - Absence of modification statement will result into infinite loop

On Entry Check - while

Syntax :

```
while (<expression>) // on entry check
 {
        ......
        //statements
 }//calls block repeatedly
```

On Entry Check - for loop

Syntax :

```
for (< initial statement > ;  < expression >  ; <Modification > )
{
        //Statements
        ......
}//calls block repeatedly
```

On Exit Check – do….while loop

- **Syntax**

```
do
{
        < Statements>
        ……
}while(<expression>);//calls block repeatedly
```

On Entry Check Vs On Exit Check

- **On Entry Check**

```
int n = 5;
While(n<=3)
{
        printf("%d",n);
}
```

- Execution control will never entered inside loop as initial state is not related to on entry expression check.

- **On Exit Check**

```
int n = 5;
do
{
        printf("%d",n);
}while(n<=3);
```

- At least one execution is fixed no matter what is initial state of expression.

Jump Statements – break, continue, return, goto

❖break

- Can be used inside switch/loop.
- Helps to move execution control forcefully outside switch/loop

❖continue

- Can be used only inside loop.
- Helps to move execution control forcefully to next iteration.
- Skips the execution of statements below continue.

❖ return

- Can be used inside function.
- Helps to move execution control forcefully back to calling function.

❖ goto :

- Syntax:

```
<label> :
        statements ;
        goto <label>;
```

- Helps to move execution control forcefully to a specific label definition

➢Function in C Programing :

- C program is made up of one or more functions
- A **function in C** is a set of statements that when called perform **some specific task.**
- It is the basic building block of a C program that provides **modularity and code reusability.**
- They are also called **subroutines or procedures** in other languages.
- C program contains at least one function i.e. main() function.
    - Execution of C program begins from main.
    - It returns exit status to the system.
- Advantages
    - Reusability
    - Readability
    - Maintainability

➢Syntax of Functions in C

- The syntax part mainly contain 3 parts as follows

1. **Function Declaration**
2. **Function Definition**
3. **Function Calls**

1. Function Declaration

- Informs compiler about **function name, argument types and return type.**
- Usually written at **the beginning of program** (source file).
- Syntax :
  - **return_type name_of_the_function (parameter_1, parameter_2);**
- Examples:
  - **float divide(int x, int y);**
  - **int fun2(int, int);**
  - **int fun3();**
  - **double fun4(void);**

Functions

- **Declaration statements are not executed at runtime**.
- The parameter name is not mandatory while declaring functions. We can also declare the function without using the name of the data variables.
  - e.g. **void demo(int , int);**

2 . Function Definition

- The function definition consists of **Block of statements** which are **executed when the function is called** (i.e. when the program control comes to the function).
- It process inputs (arguments) and produce output (return value).
- **Syntax :**

```
return_type function_name (para1_type para1_name, para2_type para2_name)
{
   // body of the function
}
```

Functions

- Function can return max one value.
- Function cannot be defined in another function.
- Example:

```
float divide(int a, int b)
{
    return (float)a/b;
}
```

3. Function Call

- A function call is a statement that instructs the compiler to execute the function. We use the function name and parameters in the function call.
- Typically function is called from other function one or more times.

Functions

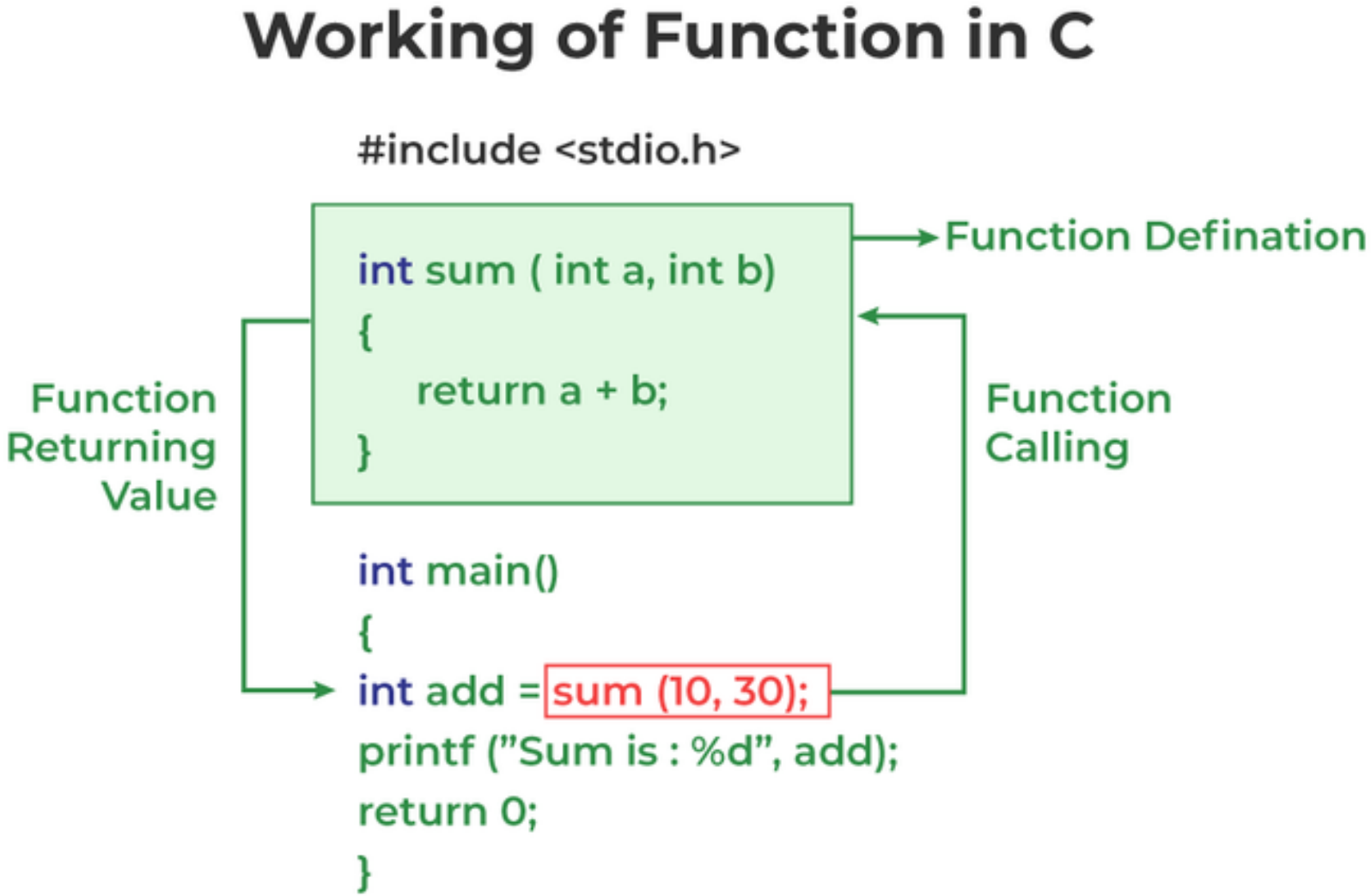
- A function can be called one or more times.
- Arguments
  - Arguments passed to function -> Actual arguments
  - Arguments collected in function -> Formal arguments
  - Formal arguments must match with actual arguments

➢Function Execution :

- When a function is called, function activation record/stack frame is created on stack of current process.
- When function is completed, function activation record is destroyed.
- Function activation record contains:
  - Local variables
  - Formal arguments
  - Return address

- Upon completion, next instruction after function call continue to execute

Function Execution:



Function Types:

- **User defined functions**
  - Declared by programmer
  - Defined by programmer
  - Called by programmer

- **Library (pre-defined) functions**
  - Declared in standard header files e.g. stdio.h, float.h, math.h, …
  - Defined in standard libraries e.g. libc.so, libm.so, …
  - Called by programmer

- **main()**
  - Entry point function – code perspective
  - User defined
  - System declared
  - int main(void) {…}
  - int main(int argc, char *argv[]) {…}

Recursion

➢ **Recursion in C**

- **Function calling itself is called as recursive function.**
- **The recursive functions contain a call to themselves somewhere in the function body.**

❖ **To write recursive function consider Some Important Points :**

- **Explain process/formula in terms of itself**
- **Decide the end/terminating condition**

Recursion

- **Syntax of Recursion**

```
   type function_name (arguments)
{
      // function statements
      // base condition
      // recursion case (recursive call)
}
```

Recursion

- **Advantages**

1. Simplified Readable programs (Divide and conquer Problems)

- **Disadvantages**

1. Needs More Space (FAR on stack)
2. Needs More Time (FAR Created)

Recursion

Loop

- Need of Modification statement else will result into infinite loop
- Less time consumption
- Less memory utilization
- Follows FIFO

Recursion

- Need of Terminating condition else will result stack overflow state
- More time consumption
- More memory utilization
- Follows LIFO

Recursion

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}
```

```
int main() {
  int res;
  res = fact(5);
  printf("%d", res);
  return 0;
}
```

5! = 5 * 4!

4! = 4 * 3!

3! = 3 * 2!

2! = 2 * 1!

1! = 1 * 0!

0! = 1

➢Storage Class in C

- A storage class in C is used to describe the following things:
  - The variable scope.
  - The location where the variable will be stored.
  - The initialized value of a variable.
  - A lifetime of variable

- A storage class represents the visibility and a location of a variable.

- It tells from what part of code we can access a variable.

Storage Classes

➢There are total four types of standard storage classes.

	Storage	Initial value	Life	Scope
auto / local	Stack	Garbage	Block	Block
register	CPU register	Garbage	Block	Block
static	Data section	Zero	Program	Limited
extern / global	Data section	Zero	Program	Program

- Each running process have following sections:
  - Text
  - Data
  - Heap
  - Stack
- Storage class decides
  - Storage (section)
  - Life (existence)
  - Scope (visibility)
- Accessing variable outside the scope raise compiler error.

Storage Classes

- **Local variables declared inside the function.**
  - Created when function is called and destroyed when function is completed.
  - All local variables by default to be considered as auto.

- **Register is similar to local storage class, but stored in CPU register for faster access.**
  - Register keyword is request to the system, which will be accepted if CPU register is available.
  - Request can be given to identify CPU register.
  - Limited Availability
  - No surety system will satisfy request
  - If request is successful will result into faster performance
  - If request unsuccessful will slow down the process . if registers are not available request will be converted implicitly to auto type
  - We can not take address of register.
  - It is allowed to request inside function.

Storage Classes

- **Static variables are same as global with limited scope.**
  - Helps to retain state of variable through multiple call of same function in which it is defined
  - Can be initialised with constant.
  - If not initialised default value is zero.
  - Static variable is initialised only once on first invocation of function providing it is initialised at the time of declaration.
  - Can have page level and function level

- **Extern / Global**
  - Applicable to global resources like variable, function, data type.
  - It describes pure declaration

Thank you!

Kiran Jaybhave

email – kiran.jaybhave@sunbeaminfo.com