SUNBEAM

WELCOMES

Sept – 2021 (PG-DAC, PG-KDAC, PG-DMC, PG-DBDA)

C Programming Revision Sessions

Trainer Introduction

- **Name**: Mrs.Akshita S.Chanchlani
- **Designation** : Technical Trainer
- **Education** :
  - PhD (Pursuing) : Computer Science and Engineering
  - Masters of Engineering (ME) in Information Technology
  - B.Tech in Computer Engineering, From VJTI Mumbai
- **Training Experience**
  - PreCAT Batches at Sunbeam
  - C, C++ , core java, python
- **Professional Experience**
  - **11+ years**
- **Email** : akshita.chanchlani@sunbeaminfo.com

Contact Admission Officers for Queries like

- Fees Related
- Documentation
- Student Activate on Sunbeam Portal (Vishal Sir , vishals@sunbeaminfo.com , 9175019069)
- MIS Registration (Rahul S Sir, rahuls@sunbeaminfo.com , 9850021565)

Point of contact for Module Software Installation related Queries

Course Name	Staff Name	Email	Mobile	Zoom Meeting ID	Password
PG-DAC and PG-KDAC	Rupesh Sutar	rupesh@sunbeaminfo.com	9923754711	8073788274	SAN
	Rupam Kapatkar	rupam.kapatkar@sunbeaminfo.com	9766825970	2157624604	SAN
PG-DMC	Sambhaji Salunkhe	sambhaji.salunkhe@sunbeaminfo.com	9021515642	9721271554	SAN
PG-DBDA	Sambhaji Salunkhe	sambhaji.salunkhe@sunbeaminfo.com	9021515642	9721271554	SAN

- **Please note, Operating System Installations are to be done by students with the help of their local hardware vendor.**
- **List of Module Wise Software will be provided by Sunbeam with Installation Guide / Support.**

Course Coordinators

Course Name	Staff Name	Mobile	Email
PG-DAC PUNE	Yogesh Sir	9921573539	yogesh.kolhe@sunbeaminfo.com
PG-DAC KARAD	Shubam Sir	8999256801	shubham.borle@sunbeaminfo.com
PG-DESD	Devendra Sir	9890662093	devendra.dhande@sunbeaminfo.com
PG-DMC	Rohan Sir	8983049388	rohan.paramane@sunbeaminfo.com
PG-DBDA	Pradnya Mam	9881736734	pradnya@sunbeaminfo.com

C PROGRAMING

by Akshita Chanchlani @ Sunbeam Infotech

Plan For C Programming

C Programming Lecture

Mrs.Akshita Chanchlani

akshita.chanchlani@sunbeaminfo.com

#9860866831

LAB

Akshita Chanchlani(9860866831)

Pradnya Mam(9623297936)

Agenda for C Programming Revision Batch

- Day1 : Functions
- Day2 : Pointer
- Day3 : 1D Array
- Day4 : 2D Array and Dynamic Memory Allocation
- Day5 : Structure

Number of Days : 5
(From 15th Sept 2021 to 20th Sept 2021)
Theory Session : 4pm to 7pm
Prime Lab Time : 11am to 12noon
*** Sunday off**

Toolchain & IDE

- Toolchain is set of tools to convert high level language program to machine level code.
  - Preprocessor
  - Compiler
  - Assembler
  - Linker
  - Debugger
  - Utilities
- Popular compiler (toolchains)
  - GCC
  - Visual Studio
- IDE – Integrated development environment
  - Visual Studio
  - Eclipse
  - VS Code (+ gcc)
  - Turbo C
  - Anjuta, KDevelop, Codeblocks, Dev C++, etc.

Software installation

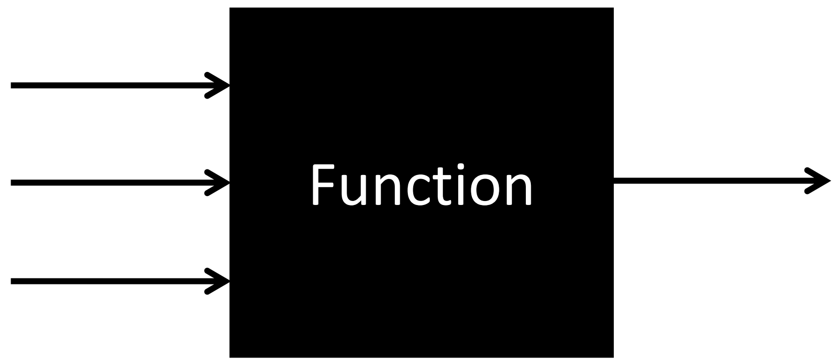
- Installations
  - GCC
  - VS Code

Day1 : Functions & Storage Classes

Functions

Functions

- C program is made up of one or more functions.
- 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

- Function is set of instructions, that takes zero or more inputs (arguments) and return result (optional).
- Function is a black box.

Functions / User Defined Functions

- It is a set of instructions written to gather as a block to complete specific functionality.
- Function can be reused.
- It is a subprogram written to reduce complexity of source code
- Function may or may not return value.
- Function may or may not take argument
- Function can return only one value at time
- Function is building block of good top-down, structured code function as a "black box“
- **Writing function helps to**
    - improve readability of source code
    - helps to reuse code
    - reduces complexity
- **Types of Functions**
    - Library Functions
    - User Defined Functions

Functions

- **Function declaration / Prototype / Function Signature**

  <return type> <functionName> ([<arg type>...]);

- **Function Definition**

  ```
  <return type> < functionName > ([<arg type> <identifier>...])
     {  // function body
     }
  ```

- **Function Call**

  <location> = < functionName >(<arg value/address>);

- 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

Examples:

1. addition()
2. print_line()
3. factorial()
4. combination()

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.
- Type and number of arguments should match the function declaration
- Return statement specifies return value, if any, and returns control to the point from which the function was invoked
- Function's return value can be ignored (statement with no assignment)
- Function cannot be defined within a function (can't nest definitions)

Passing arguments: Call by value vs Call by address/reference

- Call by value
  - Actual argument is of same type as of actual argument.
  - Actual argument is copied into formal argument.
  - Any change in actual argument does not reflect in formal argument.
  - Creating copy of argument need more space as well as time (for bigger types).
  - Most of data types can be passed by value – primitive & user defined types.

- Call by address
  - Formal argument is of pointer type (of actual argument type).
  - Address of actual argument is collected in formal argument.
  - Actual argument can be modified using formal argument.
  - To collect address only need pointer. Pointer size is same irrespective of data type.
  - Array and Functions can be passed by address only.

Storage Classes

Storage class

	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 class

- Local variables declared inside the function.
  - Created when function is called and destroyed when function is completed.

- Global variables declared outside the function.
  - Available through out the execution of program.
  - Declared using extern keyword, if not declared within scope.

- Static variables are same as global with limited scope.
  - If declared within block, limited to block scope.
  - If declared outside function, limited to file scope.

- 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.

Static

- Can be declared within function
- It is necessary to initialize static variables at the time of declaration else it violets the rule of static
- Static variable is to be initialized with constant value only.
- Static variables helps to retain state of particular variable through multiple calls of same function.
- static variables are initialized only once on first invocation of particular function in which is declared.

Register

- when register storage class is used we try to request register to identify with some name.
- There is no guarantee that our register request will be entertained.
- As number of registers availability is very less our request may be rejected and it will be automatically converted in auto type. Needs more time and slows down performance
- If request is entertained then programmer can enjoy speed/performance of application.
- we can not apply address of operator on registers.
- We can not request registers other than local scope.
- Use of register in global section is not allowed

- Syntax to declare a register variable:

```
register int regvar;
```

extern

- extern keyword extends the visibility of the C variables and C functions.
- We write **extern** keyword before a variable to tell the compiler that this variable is declared somewhere else. Basically, by writing extern keyword before any variable tells us that this variable is a global variable declared in some other program file.
- A global variable is a variable which is declared outside of all the functions. It can be accessed throughout the program and we can change its value anytime within any function through out the program.
- While declaring a global variable, some space in memory gets allocated to it like all other variables. We can assign a value to it in the same program file in which it is declared.
- **Extern** is used if we want to use it or assign it a value in any other program file.

extern

- extern keyword extends the visibility of the C variables and C functions.
- We write **extern** keyword before a variable to tell the compiler that this variable is declared somewhere else. Basically, by writing extern keyword before any variable tells us that this variable is a global variable declared in some other program file.
- A global variable is a variable which is declared outside of all the functions. It can be accessed throughout the program and we can change its value anytime within any function through out the program.
- While declaring a global variable, some space in memory gets allocated to it like all other variables. We can assign a value to it in the same program file in which it is declared.
- **Extern** is used if we want to use it or assign it a value in any other program file.

SUNBEAM

Thank you!

Akshita Chanchlani <akshita.chanchlani@sunbeaminfo.com>