# C Programming

By Dhruba Ray

# Basic Composition of a C Program

A C program is basically composed of one or more functions that are communicating with each other.

# Two Types of Functions:

- Predefined functions
- User(Programmer) defined functions

# About main:

- Prototypes of main are predefined.
- Body of main is user defined.

# Simple Standard Prototypes of the main Function

- int main() : Normally used in C++
- int main(void) : Normally used in C

# Standard Prototypes of the main Function related with Command Line Arguments

- int main(int argc, char *argv[])
- int main(int argc, char **argv)

# Standard Prototypes of the main Function related with Command Line Arguments as well as Environment variables

- int main(int argc, char *argv[], char *envp[])
- int main(int argc, char **argv, char **envp)

# Do you think the following prototype of the main function is also standard?

- main()

# What's the significance of the statement 'return 0;' ?

```
#include <stdio.h>
int main(void) {
    printf("Hello world");
    return 0;
}
```

# How many arguments are passed to the prinf function ?

```
int main(void) {
    printf("Hello world");
    return 0;
}
```

# How many arguments are passed to the prinf function in each case?

```
#include <stdio.h>
int main(void) {
    int a = 10, b = 20;
    printf("a = %d\n", a);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

# Exactly how many arguments must be passed to the printf function?

At least one.

# What is a variadic function?

- A variadic function in C can be passed any number of arguments.
- It's also know as a function with variable number of arguments.
- At least one argument of a variadic function is compulsory.

# printf is a variadic function

printf function has the following prototype:

```
int printf(const char *format_string, …);
```

# What's the significance of the sign of ellipsis(...) in the prototype of printf?

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

printf(format_string , ... );

# Do you think the following function prototype is correct?

```
void f(...);
```

# What's returned by printf function?

The number of characters successfully printed.

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
   int a = 10;
   printf("%d\n", 1 + printf("a=%d\n", a));
   return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
   int a = 10;
   printf("%d\n", 2 + printf("%d\n", 1 + printf("a=%d\n", a)));
   return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
    printf("Piss down on \
    some unsuspecting one's back \
    and tell them \
    it's raining");
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
    printf("Piss down on "
    "some unsuspecting one's back "
    "and tell them "
    "it's raining");
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
   int a = 10, b = 20;
   printf("a = %d, ""b = %i\n", a, b);
   return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
    int percent = 50;
    printf("You've got %d%%", percent);
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
    int n;
    for (n = 12345; printf("%d\n", n) - 2; n /= 10);
    return 0;
}
```

# On scanf

- Like printf scanf is also a variadic function.
- scanf has a prototyped similar to that of printf: int scanf(const char *format_string, …);
- scanf returns the total number of inputs successfully taken.
- If there are ‘n’ format specifiers within the format string of scanf then scanf can return any value in the range -1 to +n

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
   int a, b;
   printf("%d %d %d\n", a, b, 1 + scanf("%d %d", &a, &b));
   return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
   int a, b;
   printf("%d %d %d\n", 1 + scanf("%d %d", &a, &b), a, b);
   return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
   int a = 12345;
   while (printf("%d", a) + scanf("%d", &a));
   return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
    int real, imag;
    printf("Enter a complex number:\n");
    scanf("%d+i%d", &real, &imag);
    printf("(%d, %d)", real, imag);
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
    int n;
    printf("Enter a number:\n");
    scanf("%i", &n);
    printf("%d\n", n);
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
main() {
        if (printf("hello world")) {}
}
```

# What'll be the output of the following program?

```
#include <stdlib.h>
int main() {
    system("shutdown -s");
    return 0;
}
```

# Preprocessor Statemenrs

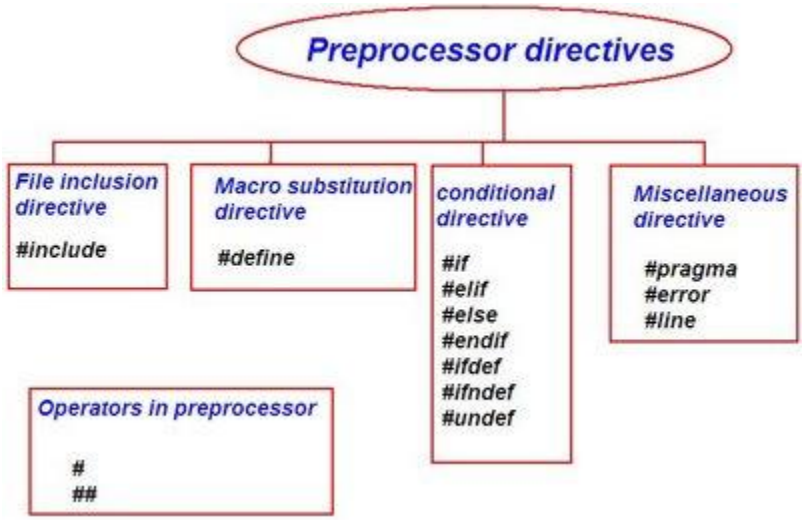
# Introduction

- Not C language statements
- Processed before compiling the code

# Construction

#Preprocessor-directive definition

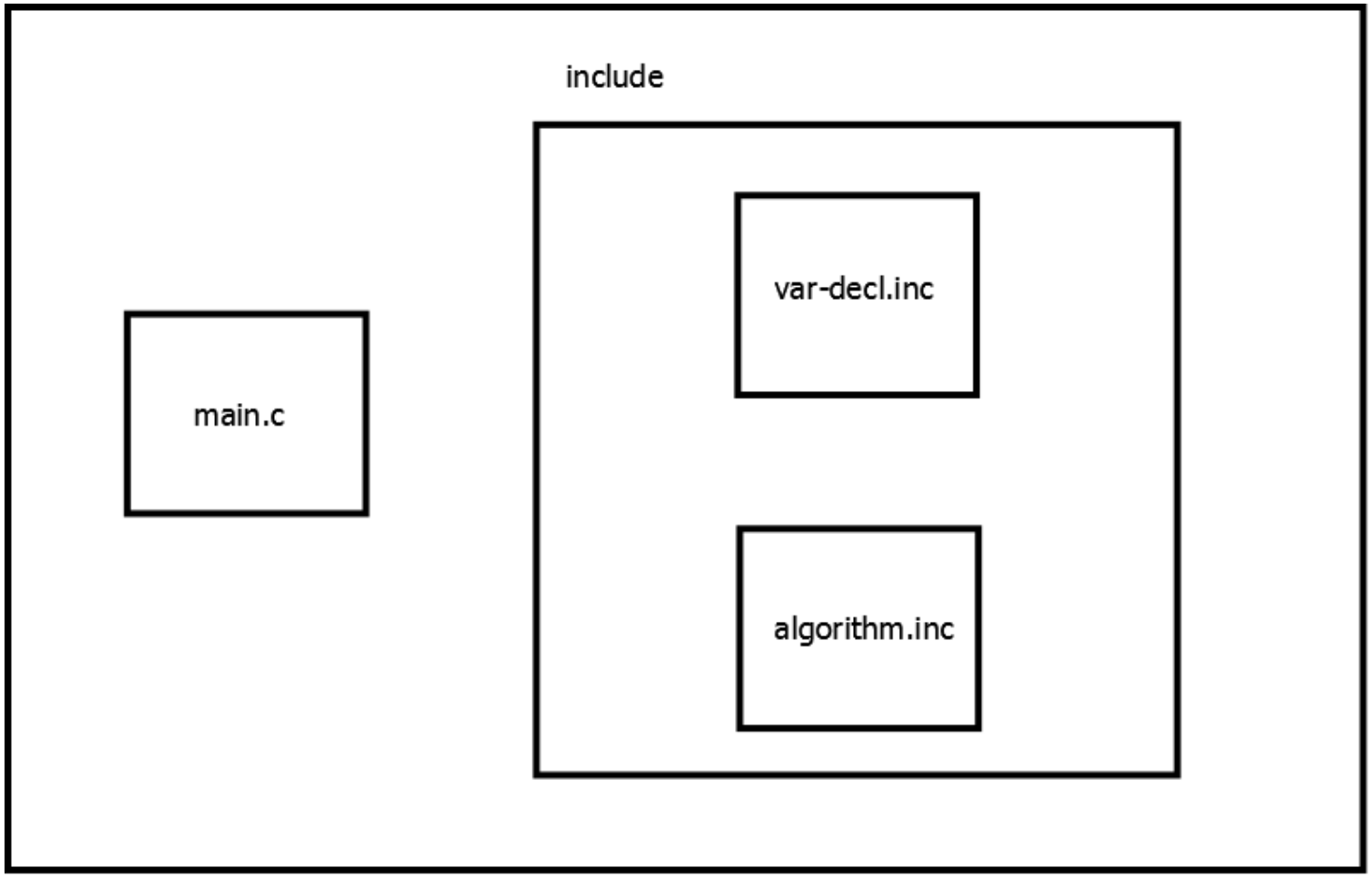
# Preprocessor Directives



# File inclusion using #include

- Difference between #include <filespec> and #include “filespec”
- Why #include <stdio.h> is preferred over #include “stdio.h”

# Problem 1: Directory structure



# Problem 1: Code of main.c

```
#include <stdio.h>

int main(void) {
    #include "include/var-decl.inc"
    #include "include/algorithm.inc"

    return 0;
}
```

# Problem 1: Code of var-decl.c

```
char *msg = "Hello World\n";
```

# Problem 1: Code of algorithm.c

```
printf("%s", msg);
```

# What'll be the output of the following program?

```
#include <stdio.h>

#define BEGIN int main() {
#define PRINT printf(
#define HELLO "Hello\n");
#define END return 0;}

BEGIN
   PRINT HELLO
END
```

# What'll be the output of the following program?

```
#include <stdio.h>
#define char_ptr1 char *
typedef char * char_ptr2;
int main(void) {
   char_ptr1 ptr1, ptr2;
   char_ptr2 ptr3, ptr4;
   printf("%d %d\n", sizeof(ptr1), sizeof(ptr2));
   printf("%d %d\n", sizeof(ptr3), sizeof(ptr4));
   return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
#define SQUARE(x) x * x
#define SQUARE2(x) (x) * (x)
#define SQUARE3(x) ((x) * (x))
int main() {
    printf("%d\n", SQUARE(5));
    printf("%d\n", SQUARE(5 + 7));
    printf("%d\n", SQUARE2(5 + 7));
    printf("%d\n", 144 / SQUARE2(5 + 7));
    printf("%d\n", 144 / SQUARE3(5 + 7));
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main() {
    int a = 1234 + printf("%d\n",
            12 + ({
                    int f = 1, i, n;
                    printf("Enter a no.\n");
                    scanf("%d", &n);
                    for (i = 2; i <= n; i++)
                            f *= i;
                    f;
            })
    );
    printf("%d\n", a);
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
#define FACTORIAL(n) ({ \
    int f = 1, i; \
    for (i = 2; i <= n; i++) \
            f *= i; \
    f; \
})
int main() {
    printf("FACTORIAL(5) = %d\n", FACTORIAL(5));
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
#define var(x, y) xy
int main(void) {
    int empsal = 10000;
    printf("%d\n", var(emp, sal));
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
#define var(x, y) x##y
int main(void) {
   int empsal = 10000;
   printf("%d\n", var(emp, sal));
   return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
#define DEBUG
int main() {
    #ifdef DEBUG
            printf("DEBUG defined\n");
    #endif
    #undef DEBUG
    #ifdef DEBUG
            printf("DEBUG defined\n");
    #endif
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
#define DEBUG
int main() {
    #if defined(DEBUG)
            printf("DEBUG defined\n");
    #else
            printf("DEBUG not defined\n");
    #endif
    #undef DEBUG
    #ifdef DEBUG
            printf("DEBUG defined\n");
    #else
            printf("DEBUG not defined\n");
    #endif
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
#define DEBUG
int main() {
    #if !defined(DEBUG)
            printf("DEBUG not defined\n");
    #else
            printf("DEBUG defined\n");
    #endif
    #undef DEBUG
    #ifndef DEBUG
            printf("DEBUG not defined\n");
    #else
            printf("DEBUG defined\n");
    #endif
    return 0;
}
```

# What'll be the output of the following program?

```
#define C
#define CPP
#ifdef C
    int main(void) {
            return 0;
    }
#elif defined (CPP)
    int main() {
            return 0;
    }
#else
    main() {
            return 0;
    }
#endif
```

# What'll be the output of the following program?

```
#include <stdio.h>
#line 100 "line-demo.c"
int main(void) {
    printf("Hello world\n");
    printf("%s\n", __FILE__);
    printf("%d\n", __LINE__);
    printf("%d\n", __LINE__);
    printf("%d\n", __LINE__);
    printf("%s\n", __DATE__);
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
#ifndef __GNUC__
   #error GCC Not defined
#endif
int main(void) {
   printf("Hello world\n");
   return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
typedef enum {
   WIN, LOSS, DRAW
} GameStatus;
#define EnumToStr(X) #X
int main(void) {
   printf("%s\n", EnumToStr(WIN));
   return 0;
}
```

# What'll be the output of the following program?

Content of main.c

```
#include <stdio.h>
int main(void) {
    #include "print.inc"
    #include "print.inc"
    return 0;
}
```

Content of print.inc

```
printf("Hello World\n");
```

# What'll be the output of the following program?

Content of main.c

```
#include <stdio.h>
int main(void) {
    #include "print.inc"
    #include "print.inc"
    return 0;
}
```

Content of print.inc

```
#pragma once
printf("Hello World\n");
```

# File Handling

By Dhruba Ray

# UNIX Classification of Files

- Regular Files
- Device Files
- Directory Files

# File Handling APIs

- Stream API
- Low level file access API in Linux Platform
- Low level file access API in WIN32 Platform

# What is a stream

- A stream is a sequence of bytes in motion

# Difference between a Stream and a String

# The standard streams

- Standard input stream
- Standard output stream
- Standard error stream

# Functions that work with Standard Streams

- printf
- scanf
- vprintf
- vscanf
- getchar
- gets
- putchar
- puts
- perror

# vprintf example

```
#include <stdio.h>
#include <stdarg.h>
void WriteFormatted ( const char * format, … ) {
    va_list args;
    va_start (args, format);
    vprintf (format, args);
     va_end (args);
}
int main () {
    WriteFormatted ("Call with %d variable argument.\n",1);
    WriteFormatted ("Call with %d variable %s.\n",2,"arguments");
    return 0;
 }
```

# vscanf example

```
#include <stdio.h>
 #include <stdarg.h>
 void GetMatches ( const char * format, … ) {
      va_list args;
      va_start (args, format);
      vscanf (format, args);
      va_end (args);
}
int main () {
      int val;
      char str[100];
      printf ("Please enter a number and a word: ");
      fflush (stdout);
      GetMatches (" %d %99s ", &val, str);
      printf ("Number read: %d\nWord read: %s\n", val, str);
      return 0;
}
```

# Data Structure representing a Stream in C

- The FILE structure
- FILE *stdout, *stdin and *stderr

# What'll be the output of the following program?

```
#include <stdio.h>


int main(void) {
   printf("sizeof (FILE): %d bytes\n", sizeof (FILE));


   return 0;
}
```

# File Handling example: File Reading

```
#include <stdio.h>
#include <stdlib.h>

int main() {
        FILE *fp;
        char c;

        fp = fopen("abc.txt", "r");

        if (fp == NULL) {
                        perror("The system cannot find the file specified.\n");
                        exit(EXIT_FAILURE);
        }

        while ((c = fgetc(fp)) != EOF)
                        fputc(c, stdout);

        fclose(fp);

        return 0;
}
```

# File opening functions

- fopen
- freopen
- tmpfile

# File opening modes

- “r”, “w”, “a”
- “rb”, “wb”, “ab”
- “r+”, “w+”, “a+”
- “rb+”, “wb+”, “ab+”

# Text File vs Binary File

# Text mode vs Binary mode

# File Handling example: A File Copy program

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
        FILE *fp1, *fp2;
        char c;
        if (argc != 3) {
                fprintf(stderr, "%s source destination\n", argv[0]);
                exit(EXIT_FAILURE);
        }
        fp1 = fopen(argv[1], "r");
        if (fp1 == NULL) {
                perror("Source not found\n");
                exit(EXIT_FAILURE);
        }
        fp2 = fopen(argv[2], "w");
        if (fp2 == NULL) {
                perror("Destination can't be created\n");
                exit(EXIT_FAILURE);
        }
        while ((c = fgetc(fp1)) != EOF)
                fputc(c, fp2);
        fclose(fp1);
```

# Redirection example 1

```
#include <stdio.h>

int main() {
	int a[10];
	int i;

	freopen("data.txt", "r", stdin);

	for (i = 0; i < 10; i++) {
		fprintf(stdout, "Enter a no.\n");
		scanf("%d", &a[i]);
	}

	for (i = 0; i < 10; i++)
		printf("%d\n", a[i]);

	return 0;
}
```

# Redirection example 2

```
#include <stdio.h>
int main () {
   freopen ("myfile.txt","w",stdout);
   printf ("This sentence is redirected to a file."); fclose
   (stdout);
   return 0;
}
```

# tmpfile() example

```
#include <stdio.h>
#include <string.h>
int main ()
{
  char buffer [256];
  FILE * pFile;
  pFile = tmpfile ();
  do {
    if (!fgets(buffer,256,stdin)) break;
    fputs (buffer,pFile);
  } while (strlen(buffer)>1);
  rewind(pFile);
  while (!feof(pFile)) {
    if (fgets (buffer,256,pFile) == NULL) break;
    fputs (buffer,stdout);
  }
  fclose (pFile);
  return 0;
}
```

# Unformatted input: fread

```
#include <stdio.h>
#include <stdlib.h>
int main () {
  FILE * pFile;
  long lSize;
  char * buffer;
  size_t result;
  pFile = fopen ( "cp.c" , "rb" );
  if (pFile==NULL) {fputs ("File error",stderr); exit (1);}
  // obtain file size:
  fseek (pFile , 0 , SEEK_END);
  lSize = ftell (pFile);
  rewind (pFile);
  // allocate memory to contain the whole file:
  buffer = (char*) malloc (sizeof(char)*lSize);
  if (buffer == NULL) {fputs ("Memory error",stderr); exit (2);}
  // copy the file into the buffer:
  result = fread (buffer,1,lSize,pFile);
  if (result != lSize) {fputs ("Reading error",stderr); exit (3);}
  /* the whole file is now loaded in the memory buffer. */
  // terminate
  fclose (pFile);
```

# Unformatted output: fwrite

```
#include <stdio.h>
int main () {
   FILE * pFile;
   char buffer[] = { 'x' , 'y' , 'z' };
   pFile = fopen ("myfile.bin", "wb");
   fwrite (buffer , sizeof(char), sizeof(buffer), pFile);
   fclose (pFile);
   return 0;
}
```

# ferror() and clearerr()

```
#include <stdio.h>
int main () {
	FILE * pFile;
	pFile = fopen("myfile.txt","r");
	if (pFile==NULL)
		perror ("Error opening file");
	else {
		fputc ('x',pFile);
		if (ferror (pFile)) {
			printf ("Error Writing to myfile.txt\n");
			clearerr (pFile);
	}
	fgetc (pFile);
	if (!ferror (pFile))
		printf ("No errors reading myfile.txt\n");
	fclose (pFile);
	}
	return 0;
}
```

# Storage Classes

# What is a storage class?

Storage class of a variable that is allocated memory space by the compiler statically has the following four features:

- Where it is allocated i.e. in RAM or CPU registers.
- Default initial value of the variable i.e. the value with which the compiler initializes the variable if the initial value is not defined the programmer.
- Scope i.e. where in your program the variable is accessible.
- Lifetime i.e. how long the variable will be alive in your program.

# Doe's every variable has a storage class?

No. If you allocate memory space for a variable dynamically then it has no storage class i.e. its fortune is not decided by the compiler: it's totally at the hand of the programmer.

# many different storage classes are there?

Exactly four: auto, register, static and extern.

# Auto:

Default for a variable declared within a block.

# Details of auto:

- Memory allocation: RAM (Stack Segment i.e. SS)
- Default initial value: not defined (i.e. garbage)
- Scope: local within the block in which it's defined.
- Lifetime: dead when the block exited.

# What will be the output of the following program?

```
int main(void) {
    auto int a = 12;

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

    return 0;
}
```

# What will be the output of the following program?

```
int main(void) {
    int a = 12;

    {
        int a = 5;
        printf("%d\n", a);
    }
    printf("%d\n", a);

    return 0;
}
```

# What will be the output of the following program?

```
int main(void) {
    int a = 12;
    int *p = &a;
    {
            int a = 5;
            printf("%d\n", a);
            printf("%d\n", *p);
    }
    printf("%d\n", a);

    return 0;
}
```

# Register:

Just a suggestion to the compiler.

# Details of register:

- Memory allocation: CPU Register
- Default initial value: not defined (i.e. garbage)
- Scope: local within the block in which it's defined.
- Lifetime: dead when the block exited.

# What will be the output of the following program?

```
int main(void) {
   register int i;

   for (i = 0; i <= 10000; i++)
         printf("%d\n", i);

   return 0;
}
```

# What will be the output of the following program?

```
int main(void) {
   register int i;

   printf("%p\n", &i);

   return 0;
}
```

# Details of static:

- Memory allocation: RAM (Data Segment: DS)
- Default initial value: 0
- Scope: local within the block in which it's defined.
- Lifetime: Alive still the program executes.

# What'll be the output of the following program?

```
int main(void) {
   int i;

   for (i = 1; i <= 5; i++) {
         int j = 10;
         printf("%d\n", j);
         j += 10;
   }
   return 0;
}
```

# What'll be the output of the following program?

```
int main(void) {
   int i;

   for (i = 1; i <= 5; i++) {
         static int j = 10;
         printf("%d\n", j);
         j += 10;
   }
   return 0;
}
```

# What'll be the output of the following program?

```
int main(void) {
    int i;

    for (i = 1; i <= 5; i++) {
            static int j;
            j = 10;
            printf("%d\n", j);
            j += 10;
    }
    return 0;
}
```

# What'll be the output of the following program?

```
int f() {
    int i = 10;
    i += 10;
    return i;
}
int main(void) {
    printf("%d\n", f());
    printf("%d\n", f());
    return 0;
}
```

# What'll be the output of the following program?

```
int f() {
    static int i = 10;
    i += 10;
    return i;
}
int main(void) {
    printf("%d\n", f());
    printf("%d\n", f());
    return 0;
}
```

# Usefulness:

Can preserve state between function calls.

# Extern:

Default for a variable declared outside the scope of any function.

# Details of Extern

- Memory allocation: RAM (initialized/uninitialized Data Segment: DS)
- Default initial value: 0
- Scope: Accessible throughout the program.
- Lifetime: Alive still the program executes.

# The role of extern keyword.

The difference between declaration and definition.

# Pointers Identity

Basic Concept

# What's a pointer?

- A pointer is an expression that represents the address of either a variable (auto in SS, static or extern in initialized/uninitialized DS or dynamically in heap) or a function (in CS or read only text segment).

# A pointer may not always appear as a variable.

- &a is a constant pointer expression

# Reading address of a variable: the address of (&) operator.

# Which is the best format specifier %d, %u or %p when you are printing a pointer using printf?

```
int main(void) {
    int a = 10;

    printf("%d\n", &a);
    printf("%u\n", &a);
    printf("%p\n", &a);

    return 0;
}
```

# Comparing size of an int and a pointer

**int**

- 16 bit: 2 bytes
- 32 bit: 4 bytes
- 64 bit 4 bytes

**Pointer(int *, double *, ...)**

- 16 bit: 2 bytes
- 32 bit: 4 bytes
- 64 bit 8 bytes

Why size of an int or a pointer varies from compiler to compiler and why not other data types?

# What'll be the output of the following program?

```
#include <stdio.h>
int main() {
   int n = 10;
   int *p = &n;
   printf("%d\n", n);
   printf("%p\n", &n);
   printf("%d\n", *&n);
    printf("%d\n", *&p);
   return 0;}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main() {
   int n = 10;
   int *p = &n;
   printf("%p\n", &*n);
   printf("%d\n", &*p);
   return 0;
}
```

# Pointer arithmetic

- Pointer + integer
- Pointer – integer
- Pointer – pointer

# Introduction to subscript operator ([])

- Let p be any pointer expression on which pointer arithmetic can be performed. The all the four following expressions are equivalent:

  p[i] = *(p + i) = *(i + p) = i[p]

# What'll be the output of the following program?

```
#include <stdio.h>
int main() {
   int n = 10;
   printf("%d\n", (&n)[0]);
   printf("%d\n", 0[&n]);
   printf("%d\n", (&n + 5)[-5]);
   printf("(-5)[&n + 5] = %d\n", (-5)[&n + 5]);
   return 0;
}
```

# Dereferencing hardcoded address

# What'll be the output of the following program? (Assume &a = 0x000000000022FE5C)

```
#include <stdio.h>
int main() {
   int n = 10;
   printf("%d\n", *(int *)0x000000000022FE5C);
   printf("%d\n", 0[(int *)0x000000000022FE5C]);
   return 0;
}
```

# void pointers (Example 1)

```
#include <stdio.h>
int main(void) {
    int a = 10;
    void *pa = &a;
    void **ppa = &pa;
    // printf("%d", (int)*pa); error
    printf("%d", *(int *)pa);
    // printf("%d", **ppa); error
    printf("%d", *(int *)*ppa);
    printf("%d", **(int **)ppa);
    return 0;
}
```

# void pointers (Example 2)

```
#include <stdio.h>
int main(void) {
   int a = 10;
   void *pa = &a;
   void *pa2 = &pa;
   printf("%d", **(int **)pa2);
   printf("%d", *(int *)*(void **)pa2);
   return 0;
}
```

# Examples of standard library functions working with void pointers: Dynamic memory allocation

- void* malloc(size_t size);
- void free(void* ptr);
- void *calloc(size_t num, size_t size);
- void *realloc(void *ptr, size_t size);

# DMA example 1

```
#include <stdio.h>
int main(void) {
    int *p;
    int n;
    printf("Enter no. of ints\n");
    scanf("%d", &n);
    p = (int *)malloc(n * sizeof (int));
    for (i = 0; i < n; i++)
            printf("%d", p[i]));
    free(p);
}
```

# DMA example 2

```
#include <stdio.h>
int main(void) {
    int *p;
    int n;
    printf("Enter no. of ints\n");
    scanf("%d", &n);
    p = (int *)calloc(n, sizeof (int));
    for (i = 0; i < n; i++)
            printf("%d", p[i]));
    free(p);
}
```

# Constant pointers: Example 1

```
int a = 10;

const int *p1 = &a;

int const *p2 = &a;

int * const p3 = &a;

const int * const p4 = &a;
```

# Constant pointers: Example 2

```
#include <stdio.h>
int main() {
   int a = 5;
   int  const * p=&a;
   printf("%d",++(*p));
   return 0;
}
```

# Pointers Supremacy

Arrays and Pointers

# Array Definition and initialization

- int a[] = {12, 5, 7, 13, 6};
- int a[10] = {12, 5, 7, 13, 6};
- int a[] = {[2]: 2, [5]: 5};
- int n = 10; int a[n];

# Accessing Array Elements Example 1

```
#include <stdio.h>
int main(void) {
    int a[] = {12, 5, 7, 13, 6};
    int i;
    for (i = 0; i < 5; i++)
            printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

# Accessing Array Elements Example 2

```
#include <stdio.h>
#define SIZE 5
int main(void) {
    int a[] = {12, 5, 7, 13, 6};
    int i;
    for (i = 0; i < SIZE; i++)
            printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

# Accessing Array Elements Example 2

```
#include <stdio.h>

int main(void) {
    int a[] = {12, 5, 7, 13, 6};
    int n = sizeof(a) / sizeof(a[0]);
    int i;
    for (i = 0; i < n; i++)
            printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

# Which one of the following statements is incorrect and why?

```
int a[5];
int *p = a;
 int **pp = &p;
 int **pa = &a;
```

# What's the difference between the following declarations?

- int *p[10] and
- int (*q)[10]

# Concept of C declarators

```
int a[5] = {17, 5, 8, 98, 6};
int *b[5] = {a, a + 1, a + 2, a + 3, a + 4};
int *p1 = a;
int (*p3)[5] = &a;
int *(*p4)[5] = &b;
int (**p5)[5] = &p3;
int *(**p6)[5] = &p4;
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main(void) {
    int i;
    int a[] = {12, 5, 7, 13, 6};
    int (*pa)[] = &a; // pointer 2 an array of integers
    int *pa2[] = {a, a + 1, a + 2, a + 3, a + 4}; // array of pointers 2 integers
    for (i = 0; i < 5; i++)
            printf("%d\n", *(*pa + i));
    for (i = 0; i < 5; i++)
            printf("%d\n", *pa2[i]);
    for (i = 0; i < 5; i++)
            printf("%p\n", *(pa2 + i));
    for (i = 0; i < 5; i++)
            printf("%d\n", **(pa2 + i));
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int main() {
    int a[5] = {1};
    int b[5] = {2};
    int c[5] = {3};
    int d[5] = {4};
    int e[5] = {5};
    int (*v[])[5] = {&a, &b, &c, &d, &e};
    int i;
    for (i = 0; i < 5; i++) {
            for (j = 0; j < 5; j++)
                    printf("%d ", (*v[i])[j]);
            printf("\n");
    }
    return 0;
}
```

# What'll be the output of the following program?

```
#include <stdio.h>
int (*changeValue(int (*pa)[], int i, int n))[] {
    (*pa)[i] = n;
    return pa;
}
int main(void) {
    int a[] = {10, 20, 30, 40, 50};
    int i;
    3[*changeValue(&a, 2, 60)] = 70;
    for (i = 0; i < 5; i++)
            printf("%d ", a[i]);
    return 0;
}
```

# Pointers Ultimatum

Function Pointers

# Union

# A Program to show whether a system is BIG ENDIAN or LITTLE ENDIAN

```
union {
    short s;
    char c[2];
} u;
int main(void) {
    u.s = 0x0102;
    if (u.c[0] == 1 && u.c[1] == 2)
            printf("BIG ENDIAN");
    else
            printf("LITTLE ENDIAN");
    return 0;
}
```

# Functions

# Variadic function: Header file to be included

```
#include <stdarg.h>
```

# An example of a variadic function

```
#include <stdarg.h>
#include <stdio.h>
double average ( int num, …) {
   va_list arguments;
   double sum = 0;
     int x;
va_start ( arguments, num );
for (x = 0; x < num; x++ ) {
   sum += va_arg( arguments, double );
 }
   va_end ( arguments );
   return sum / num;
}
int main() {
   printf( "%lf\n", average ( 3, 12.2, 22.3, 4.5 ) );
   printf( "%lf\n", average ( 5, 3.3, 2.2, 1.1, 5.5, 3.3 ) );
     return 0;
}
```