Week 1:

SDLC – Soft development life cycle (business process), also known as Waterfall.

Requirements Analysis (Change features) > before coding they will **Design** (Architecture is the most difficult part to change) > **Development** > **Testing** > **Deployment** (uploading) > **Maintenance** (feedback from consumers)

Requirements Analysis > Design > Development > Testing > Deployment > Maintenance

Cd: c:/cygwin64

Key types of Operations performed by CPU:

- Copying data
- Mathematical functions (+ - x divide)
- Comparisons
- Jump Instructions
-

Registers consists of:

IP - Instruction pointer

PC - Program Counter

LFES - Load Fetch Execute Save life cycle

Machine codes - Binary representation of instructions

Assemblers / Assembly languages – Human readable representation of instructions as understood by CPU (dependent on the instructions set (ISA)

High-level languages – represent platform independent statements that represents business logic that is important to be translated into machines codes

Lesson 2: C language:

(.C) > Pre-processor (.C) > Compiler (.C) > (.asm) > Assembler > (.

Input A > Read A > Input B > Read B > Add A + B > Output "Result is" C > End

Notepad++

```
#include <stdio.h> //pre processor // angle bracket = system

#include "other.h" //There is another file called other.h

Int main (void)  //CRLF (Character return line fill)

{

Printf("Hello World\r\n");          // r\n end of line(?)   or \n

Return 0;

}
```

```
#include <stdio.h> /*printf, scanf*/

Int add(int n1, int n2)

{

        int result

        result = n1 + n2

        return result;

}

int main(void)

{

        int a;

        int b;

        int c;


        Printf("Input A: ")

        scanf("%d", &a)                                                    //passsby copy

        /*reading A*/

        Printf("Input B: ")

        scanf("%d", &b)

        /*reading B*/

        return 0;


        C = add(a,b)

}
```

Week 2:

1. Compiled (edit > compiler > link > test)
- Source > processor > compiler > assembler > link > executable (Slow and takes time)
- Source Code : Translation Units (*. c) or Headers (*.h)

2. Interpreted source > interpreter > run > interpreter (Cycle – Virtual Machine)

3. Mixed approach (JIT):
   - Source > Basic Compilation > integration > Run (Fast, Run as long as u get the source)

Sequences of bytes > logical analysis > - (Token) -> System Analysis > - (Parse Tree) > Semantic Analysis – (Abstract System tree)> Optimisation & Code Generation > Assembly Code

Int a; =/= a int;

Assign a real number with fractions into integer, will have error too.

Command prompt:

```
Cd c:\cpp

C:\cpp>mkdir lecture 2019010

C:\cpp>mkdir lecture 2019010 >echo.>main.c
```

C89 cppreference printf

```
int main (void)

{

Int I;

Char c;

Int numRead;

numRead = scanf("%i %c, &I, &c);

printf(You have entered:

C:\c>gcc – Wall – Wextra  -Wconversion –Werror –ansi – pedantic –o main.exe main.c
```

Week 3: Statement and expressions

Expressions – represents values

- Literals (10, 0x04ULL)
- Variables (int I =10; i = 20;)
- Constants (constant float PI = 3.14F;)
- Compounded expressions
- Unary operators (e.g. –i)
- Binary operators (i – i)
- Tenary operator (x = c?a:b);

```
Left Operand + right operand        //binary expression

i = (i    +    A)                   2
```

Selection

- If (condition) statement
    - 0 (false), !0 (true)
    - Int i =10;
    - If (i)
      {
      Printf("non-zero!");
      }

- If else (condition) statement
    - If (condition)
      {
      Statement 1;
      }
      Else
      {
      Statement 2;
      }

- If else if (condition) Statement
- If (c1)
  {
  Statement 1;
  }
  Else if (c2)
  {
  Statement 2;
  }
  Else if (c3)

```
    {
    Statement 3;
    }
    Else
    {
    Statement 4;
    }
```

<u>Switch (expression):</u>

```
{

Case 0 : statement 0;

Case 1: statement 1;

Case 2: statement 2;

Default: statementDef;

}
```

<u>Switch (expression):</u>

```
{
Case 0: statement 1;
        Break;
Case 1: statement 2;
        Break;
Default: statementDef;
        Break;
```

- Ternary operator

```
X =C?a:b;

If (C)

X = a;

Else

X = b;
```

- Nested if else loop
- if(c1)

```
    {
    If (c2)
    {
    Statement 2;
```

```
}
Else
{
Statement 3;
}
}
```

- Iteration
  - While (cond)
    Statement;

```
Int i = 0;
While (i <10)
i = i + 1;

while (0)                /*executes 0 times */
statement;

While (1)                /*an infinite loop*/
Statement;
```

- Do while (condition) statement
  Do
  Statement;
  While (condition);

- For loop (initialisation, condition expression, progression expression) statement

```
#include <stdio.h>

Int main (void)

{

Int i;

For (i=0; i<10, i++)

Printf("%d", i);


For (i=9, i>=0, i--)

Printf("%d", i);

}
```

- Jump
  - Break;
    /*jump to the end of the switch/case, jump out of loop (while/do while/ for loop */

- Continue;
- Goto /* Don't use it, unless u pro */

```
Label: 1=i+1;
I=i+2;
Goto label1;
```

- Return /* sets the return value of a function */
  /* stops the execution of a function*/

```
#include <stdio.h>


int main( void)
{
Int i;
Int j;

Scanf("%d", &i);
Scanf("%d", &j);

If(j)
Printf("%d", i/j);
Else
{
Printf("Division by 0!"), return 1;
}
/*                    */
Return 0;
}
```

- Compound statements

```
Statement
{
Statement 1;
Statement 2:
}


#include <stdio.h>
Int main (void)
{
```

```
Int i;

For (i=0; i<10, i++)

{

If (i%3)                                        /* % is remainder */

{

printf ("%d, i);

}

Else

{

Printf("!");

}

Return 0;
```

Classification of expressions:

- Based on number of operands
  - Unary

    1. Operand (value), operator (value)
    e.g. i ++;

    2. Operator operand
    - 1 -i

- Binary
  - Operand operator operand
    e.g i + 1;
    e.g 7%count;

- Ternary
  - x = condition?a:b;
  - based on their priority
    - based on the operator precedence
    - based on the associativity of an operator
      - Left to right e.g. a+b+c; > (a+b) +c; or j=i=1;

How data types impact machine codes?

- After compilation, variables have no names. (Just addresses) and no types.
- A single operator can be translated to different low level instruction depending on arguments.

/*integers*/ I = 1+2:

/*floats*/ f= 1.0f +2.0f;

I=1/2; /* for i=5 */

F = f/2;

- Operators
  - Arithmentic
    - Unary
      - +a , -a , ~a (bitwise)
    - Binary
      - A+b
      - A-b
      - A*b
      - A/b
        - e.g. 1/2 = 0
        - e.g. 1/ (float) 2 = 0.5
        - a%b
    - SHL (shift left – binary)
      - a<<b
    - SHL (shift right – binary)
      - 0000 11111 > shift by 2 > 0x0F (hexadecimal)

<u>Pre / post: (increment/decreament)</u>

- Unary
- {++ i;} or {-- i;} /* increase or decrease I and return new value */
- {i++;} or {i--}; /* set the result to i, increase I by 1, return the old value (result)
  - t=i++;
    t=i;
    i++;
- logical
  - unary
    - !a
  - Binary
    - A ||b
    - A && b
- Assignment operators
- A=b;
- Compound assignment
  - E.g. i=i+10; equals i+=10; (same but shorter form(
  - E.g. i=i-10; equals i-=10;
  - i=i*10; equals i* = 10;

- i=i/10; equals i/=10;
- i=i%10; equals i%=10;
- i=i&10; equals i&=10;
- i=i|10; equals i|=10;
- i=i^10; equals i^=10;
- i=i<<10; equals i<<=10;
- i=i>>10; equals i>>=10;

*Others

e.g. a[b] , *a, &a , a->b, a.b , a(p1,p2)pr

```
#include <stdio.h>

Int main(void)

{

Int a;

Int b;

Int c;


Char c;

Printf("Write down an expression: ");

Scanf(%d,%c,%d, &a, &c, &b)

Return 0;

}
```

<u>Week 4:</u>

1. Arithemetic / bitwise
2. Assignment
3. Logic
4. Inc/Dec
5. Comparison
6. Others

```
#include <stdio.h>

Int isDivBy5(int value)
{

Int result;

Return (value %5==0);

If(result)

{

printf("Divisible");

Else

{

printf("Not Divisible");

}


Int main (void)

{

Int a = 5;

Int b = 7;

If ((a!=b_ && (isDivBy5(a)))

{

Printf("Special");

Else

Printf("Not Special);


# include <stdio.h>
```

```
Int main (void)

{

Int input;

Printf(“Enter a positive number: “);

If (scanf(“%d”, &value)==1)

            {

            If (input > 0)

            {

            Printf(“Good Job!”);

            }
            else
            {

            Printf(“You have failed me”);

            }

            }

            Else

            {

            Printf(“You should have input a positive number.”);

            }

            Return 0;

}


# include <stdio.h>

Int main (void)

{

Int input;

Printf(“Enter a positive number: “);

If (scanf(“%d”, &value)!=1)

            {

            If (input > 0)

            {

            Printf(“You should have input a positive number”);
```

```
        }
        else if (input >0)

        {

        Printf("Good Job");

        }

        Else

        {

        Printf("You have failed me");

        }

        Return 0;

}


# include <stdio.h>

Int main (void)

{

Int input;

printf("Enter a positive number: ");

If (scanf("%d", &value)==1) && (Input > 0))

        {

        do

        {

        Printf("*");

        --input;
        }
        While (input! =0);

        Return 0;

}
```

```
gcc – Wall –Werror –Wextra –Wconversion –ansi –pedantic –o main1.exe main1.c
```

```
# include <stdio.h>

Int main (void)
```

```
{
Int input;
printf(“Enter a positive number: “);
If (scanf(“%d”, &value)==1) && (Input > 0))
        {
        while(input)
        {
        printf(“*”);
        --input;
        }
        Return 0;
}

# include <stdio.h>
Int main (void)
{
Int counter;
Int input;
printf(“Enter a positive number: “);
If (scanf(“%d”, &value)==1) && (Input > 0))
        {
        For (counter=0; count< input; ++counter)
        {
                If (counter <5)
                {
                Break;
                }
                printf(“*”)
        }
        For (;input;--input)
        {
```

```
        printf(“*”);
        }
        Return 0;
}

#include <stdio.h>
Int main (void)
{
unsigned int n;
printf(“Please input a number from 1 to 4: “);
if(scanf(“%d”, &n) ==1) && (n >=1 && (n <= 4))
{
Printf(“*”)
}
Return 0;
}

#include <stdio.h>
Int main (void)
{
unsigned int n;
unsigned rowIndex;
unsigned int colIndex;
unsigned int lvlIndex;
printf(“Please input a number from 1 to 4: “);
if(scanf(“%d”, &n) ==1) && (n >=1 && (n <= 4))
{
        For (lvlIndex =0;LevelIndex<n; ++lvlIndex)
        {
        For(rowIndex =0;rowIndex <n; ++rowIndex)
        {
        printf(“*”);
        colIndex =0;
        while (colIndex < row Index)
        {
        Printf(“**”);
        ++colIndex;
        }
        printf(“\n”);
}
}
Return 0;
}

#include <stdio.h>
const float EPSILON =1e-3f;                                        //1e-3 = 1000
Int main (void)
{
        Float n;
        printf(“Input a number greater or equal to 1: “);
        if(scanf(%f, &n)!=1) || (n <1.0f))
        {
        Printf(“Better luck next time!”);
        Return 1;
        }
        a=1.0f;
        b= n;
        do
        {
        c = (a+b)/2;
        if (c*c-n < EPSILON)                       //3-4 <1/1000
        break;
        Else if(c*c>n)
        b=c;
        Else
        a=c;
        }while(1);
printf("SQRT: %f", &c );
}
```

Week 5:

Source code:

.c (compile) > Pre-processor (translation unit) > compiler > assembly code > object file > linker

.h (header)> Pre-processor (translation unit) > compiler >assembly code > object file > linker

Header
.Text
.Data (!=0 will be store here)
.Read only data (constant)
DSS
Heap
Stack

1. Copies executables to new memory space
2. Allocate additional segments (DSS, Stack)
3. What is left is a heap.
4. OS creates a process (mem + resources needs to execute the program)
5. OS create the main thread in the process
6. OS starts the thread execution from main ()

```
Int function()

{

Int a = 0;

Return 1;

}
```

Step to call a function [int function()]:

1. Remember the next instruction that should be executed when the function returns.
2. Remember the actual parameters of the function call.
3. Allocate memory for the result.
4. Jump to the first instruction of the function.
5. Execute statements until return.
6. Deallocate no.2
7. Jump to no.1
8. Copy no.3 to other memory if needed. (Resume main)

1. **Storage:**

- Automatic storage (local variables, function's formal parameters)
- Static storage (global variables, keywords: static)
- External storage (keyword: extern)

- Register storage

2. Scope (Visibility)

- Block/ local scope (to the end of the block/scope)
- Global scope (to the end of the translation unit)
- Function scope (labels for goto)
- Function prototype scope (declaration.only)

```
#include <stdio.h>

Int res;

Int nom;

Void divide (int nom, int den)

{

Res = nom / den;

Rem = nom % den;

}

Int main(void)

{

        Divide (100,13);

        Printf("%d,%d , res , rem  );

        Return 0;

}
```

Output: 7, 9

```
#include <stdio.h>


Int main(void)

{

        Int a=5;

        Label1:

        If (a>3)

        {
                --a;
                Goto label1;
        }
        Label1:
        return 0;
}

#include <stdio.h>
Void f()
{
        Int a=0;
        ++a;
        Printf("%d", a);
        Return;
}
Int main (void)
{
        F();
        Return 0;
}

#include <stdio.h>
Void printNext(int c, int r);
Int main(void)
{
int r, c;
for (r=0; r<=10;++r)
{
        For (c =0; c<=10;c++)
        {
        Printf("%4i",r*c);
```

```
          }
          Printf("\n");
}
return 0;
void getNext (int c, int r)
{
Printf("%4i", r*c);
```

Week 6:

8bits: char, unsigned char, signed char

16bits: short int, unsigned short int

16-32: bits Int, unsigned int

>32 bits: Long int, unsigned long int, long long int

Unsigned representation:

Unsigned short int

0000 0010 0000 1111 = 512 + 8 + 4 + 2 + 1 = 527

1000 0000 0000 0001 = 32768+1

Signed short int

0000 0010 0000 1111 = 512 + 15 = 527

1000 0000 0000 0001 = 1+(-32728) = -32767

Real

Float(32)

Double(64)

Long double

i = i +1;

s = i + 1;

- Usual type conversion(implicit)
- User defined (explicit) conversion

f = i +1;

i = f + 1;

case 1: if any of the operands is a real number (float, double, long double), convert the other operand to real number.

Case 2: if not case 1; increment the size of the smaller operand to the bigger one. (promotion):

Unsigned int > upgrade > signed int > upgrade > unsigned long int > upgrade > signed long integer > upgrade > unsigned long long int > signed long long int

Unsigned char = unsigned char + unsigned short int

When are the implicit conversions being used?

- Within expressions using operands

- when calling functions

```
Void f(int);

F(uc);                //When returning a value from a function


Int f(void)

{

Return uc:

}
```

- Numberic overflow
  - A situation where are result of an expression is greater than the available sorage of a data type.

```
Unsigned char c = 255;
1111 1111
c+c+1;  /*well defined, result = 0*/

signed char c = 127;
c = 127 + 1;       /* on Intel x86 -128; undefined behaviour */

0111 1111
1000 0000

#include <stdio.h>
Int main(void)
{
Unsigned short int us = 40000;
Signed short int s = us;
Return 0;

Printf("%x", s);
```

<u>Arrays</u>

//A continuous block of memory of N elements each of a given type

//static array – the number of elements, the size is a compile –time constant

```
//dynamic array (after midterm)

Int main(void)
{
Int numbers[4];
Int I;
For(i=0;i<4; ++i)
{
scanf("%d", &number[i]);
}
While(i>0)
{
--I;
Printf("%d\n", number[i]);

Return 0;

Typedef int Element;                    //?????????????????????
```

Arrays:

```
#include <stdio.h>
Long unsigned int size_arr(double darr[10], long unsigned int size)
{
        Return sizeof(darr)/sizeof(darr[0]);
}
Int main(void)
{
        Double darr[10];
        Double darr[20];
        Printf("%lu", sizeof(darr)/sizeof(darr[0]);          //length of array: ans: 10
        Return 0;
```

```
}

#include <stdio.h>
Void setElement(int c, int r, int ns[], int cos, int value)
{
        Int I;
        I=r*cols+c;
        ns[i]=value;
setElement(2, 1, numbers, COLS, 123);
Int main(void)
{
        #define ROWS 2
        #define COLS 5
        #define SIZE ROW*COLS

        Int I, c, r;
        Int numbers[SIZE] ={0};

        For(r=0;r<ROWS; ++r)
        {
                For(c=0;c<COLS; ++c)
                {
                I=r*COLS+c;
                Printf("%d ", numbers[i]);
                }
                Printf("\n");

        For(i=0; i<SIZE; ++i)
        {
        Numbers[i]=I;
        }
```

```
            For(i=0;i<SIZE;++i)
            {
            Printf("%d ", numbers[i]);
            }
            Return 0;
}
Printf("\n-------------------\n");
I=SIZE;
While(i>0)
{
            --I;
            Printf("%d ",number[i]);
}
```

POINTERS

-Pointers to a value of unknown type: void

Uses cases:

- We want to work on an object, not its copy.
- We want functions to pass-by-copy a point to original object, not a copy of the object itself.
- Point to something in memory.
- Point to something in memory.
- We want to store the address of a dynamically allocated memory.

Declaration of a variable:

Unsigned long long* ullp;

Typedef unsigned long long* pull;

Pull a,b;

Operations

Assignments:

Int *a,*b

a=b;

getting the addres of an object

int*pi;

int I = 10;

pi =*I;

de-reference(using the pointer an the pointed object)

int*pi;

int I = 20;

pi=&I;

(*pi)=20;

Comparision:

!=

==

Pointer arithmetics

++

--

P+=offset;

p-=offset;

p=p+S;

void*vp=0X0012;

vp+=2; /*0x0014*/

int* ip = 0x0112;

ip+=2;

#define NULL 0

```
#define (void*)0

Int*ip=NULL;

If(ip!=NULL){}


Int*pi=NULL

Int I = 20;

Pi=&I;

*pi=40;
```

Problems related to pointers:

-dereferencing a NULL pointer

```
Int*pi=NULL;

*pi=10;
```

-dereference an unassigned pointer:

```
Int*pi;

*pi=20;
```

-memory leak – acquired is not released.

Double delete – acquired memory is released and released again(?);

Dangling points – a pointer points to a memory location that is not allocated, o is allocated for an unrelated object.

```
#include <stdio.h>

Int*find_greatest(int*pa,int*pb, int*pc)

{

        If(*pa>*pb)

        {

                If(*pc>*pa)

                {

                Return pc;
                }
                Else
                {
                Return pa;
        }
        Else
        {
                If(*pc>*pb)
                {
                Return pc;
                }
                Else
                {
                Return result = pb;
                }
        }

        #include <stdio.h>
        Double*get_ptr()
        {
                Double d;
                Return &d;       //don't return address of variables
        }
        Int main(void)
        {
        Printf("Address is:%p", get_ptr(20))
        Return 0;


}
Int main(void)
```

```
{
            Int*greatest_pt=NULL;
            Int a,b,c;
            Printf("enter a,b,c\n");
            Scanf("%d,%d,%d,&a,&b,&c);
            Greatest_ptr = find_greatest(&a,&b,&c)

If(a>b)
{
            If(c>a)
            {
            Greatest_pt&c:
            }
            Else
            {
             greatest_pt&a;
            }



==================================================================================
#include <stdio.h>

Void swap(float*a, float *b)
{
            Float temp = *a;
            *a = *b;
            *b = temp;
}

Int main (void)
{
	Float x, y;
	Printf(“Enter two numbers: “);
	Scanf(“%f %f, &x, &y);

	Swap(&x, &y);
	Printf(“%f %f, x, y);
	Return 0;
}
================================================================================
#include <stdio.h>

Int main(void)
{
	float myfloat = 1.5f;
	float my2ndfloat = 2.5f;

	print(&myfloat);
	print(&my2ndfloat);
	printf(“%f”, myfloat);

	return 0;
}

Void print(const float* const f)
{
	If (*f=0.0f);
	{
	printf(“%f\n”,*f );
	}
}
```

Notes:

Only applies to pointers:

1. If **const** goes before the type; I have a non-const pointer to a const type
- E.g const float* is a pointer to a constant float (“low-level constness”)

2. If const goes after the type: I have a const pointer to a non-const type
- E.g. float*const is a const pointer to a non-const float (“high-level constness”)

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


Void print(cont unsigned int* a, unsigned int length)

{

            Unsigned int I;

            For (i = 0; i<length; ++i)

            {

            Printf(“%u “, *(a+i));

            Printf(“%u “, a[i]);                    //comment away

            Printf(“\n”);

Int main(void)

{

            unsigned int I;

            unsigned int a[10];


            srand((unsigned int)time(NULL));


            for(i=0; i<10; ++i)

            {

            a[i]=(unsigned int)rand();                    //RAND_MAX

            }
            Printf(“\n”);
            /*--------program code --------*/
            Print(a, sizeof(a)/sizeof(a[0]));
            /*----------end code ------------*/

            For (i=0; I<10;++i)
            {
            printf(“%u “, *(a+i));
            }

            return 0;
}
```

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

Void swap(float*const a, float*const b)
{
            Float t = *a;
            *a = *b;
            *b = t;
}
Void Sort(float* const array, const unsigned int length)
{
            Unsigned int I,j;
            For (i=0; i<length; ++i)
            {
                        For (j=0; j <length; ++i)
                        {
                                    If( I !=j && (array[i] >array[j]))
                        {
                        Swap(&array[i], &array[j]);        //comment away
                        Swap (array +i, array + js)
                        }
                }
        }
}

Void print(const float* const array, const unsigned int length)
{
        Unsigned int I;
        For (I =0; i< length; ++i)
        {
                Printf("%.3f ", array[i]);
        }
        printf("\n");
}

Int main(void)
{
        #define LENGTH 20
        float arr[LENGTH];
        Unsigned int i;
        Srand((unsigned int)time(NULL));

        For (i=0; i<LENGTH; ++i)
        {
                Arr[i] = (float) rand()%100);
        }
        Print(arr, LENGTH);
        Sort(arr, LENGTH);
        Print(arr, LENGTH);

        Return 0;

}
```

```
#include <stdio.h>

Int main(void)

{

        Typedef float row[4];

        Row arr[3] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};

        Row arr[3] = {1,2,3,4,5,6,7,8,9,10,11,12};          //is the same

        (void)arr;


        Printf("Size of arr:%lu", sizeof (arr));


        For(i=0; i<3;++i)

        {

                For(j=0; j<4; ++j)

                {

                        Printf("%f ", arra[i][j])

                }

                Printf("\n");

        }

//float arr[3][4]

//array will decay to float(*)[4]

//float x[4];

//arr[0] will decay to float*

Printf("My float: %f", arr[0]);

        Return 0;

}
```

```
#include <stdio.h>
Int main(void)
{
        Char*s = “ABC”;              //ASCII encoding
        printf(“%s”, s);

        Return 0;                    //Print ABC
}
```

String literals

Header:

Start execution at the address of the first op-code of the main functions.

When the program is loaded, allocate a call stack of the size S.

When the program is loaded, allocate a BSS segment of the size B.

.Text:

All the code from you program (this includes our main(), but also other included functions, such as printf()

.Data

All static variables initialized with non-zero values.

.ROData

All static constants and some of the literals (esp. strings)

“ABC\0”

.BSS

All static variables initialized with zero values.

Unallocated segments we refer to as a heap

.Stack

```
#include <stdio.h>
Int main(void)
{
	Char*prompt = “Input a string”;
	Printf(“%s”,prompt);

	char arr[8] = {‘A’,’B’,’C’};		//”ABC”;
	arr[o] = ‘D’;
	*prompt = ‘0’;

	printf(prompt);
	printf(arr);
	Return 0;
}
```

```
#int main(void)
{
	Char str[8] = {0};
	Char c;
	int i =0;

	Do
	{
		Scanf(“%c”,&c);
		If(c == ‘\n’)
		{
			Break;
		}
```

```
                Str[i++] = c;

                If(i==7)

                {

                        Break;

                }

        }while (1)

        Prinf("Your string is %s", str);

        Return 0;

}
```

==

```
#include <stdio.h>

Int main (void)

{

        Char str[8] ={0};

        Scanf("%7s",str);

        Printf("Your string is:",str);

        Return 0;

}


Int main (void)

{

        Char str[8] ={0};

        Scanf("%7s",str);

        Printf("Your string is:",str);

        Return 0;

}
```

```
Int main (void)

{

        Char a = "ABC";
        Char b = a+2;                           // char b = "c"
}
```

.RODATA

(1)

ABC0C0

^a ^b

(2)

ABC0

^a^b

```
#include <stdio.h>

Size_t strlength(char*s)
{
Size_t result = 0;
While(*s)
{
        s++;
        result++;
}
}

Int main(void)
{
        Char*s = "ABCD";
        printf("Size in memory: ", sizeof(s));
        printf("Length of string: ",strLen());
```

A string literal will end up in a special session

Create an array of bites and read the input of the user but have to limit the input.

```
#include <stdio.h>
Int main(void)
{
        FILE*file;
        (void) file;
        Fopen(“myfile.txt, “r”)            // r, w, a, r+,w+, a+,rb,wb,ab,w+b,wb+
        If(file == NULL)
        {
                Printf(“Could not open a file!”);
        }
        else
        {
                 fprintf(file, ”Hello World\r\n”);    // Hello World in txt file in writing mode
                fprintf(file, “Hello!”);                  // Hello World Hello! In a+ (Append+) mode
                Printf(“Open a file successfully!”)
        }
        Return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
Int main (int argc, char** argv)              //*int main(void)
{
        /****************************
        Int I;
```

```
printf(“%i”, argc);
printf(“%s”,argv[0]);
if(argc>1)
{
        Printf(%s”, argv[1]);
}
(void) argv;
 ****************************/
FILE*file;
Char filename [255];
Printf(“Provide a file name”);
If(scanf(“%254s”, filename)!=1)
{
Printf(“Wrong number of arguments!”);
Exit(2);               //exit(1);
}


(void) file;
File = fopen(argv[i], “r”)          // r, w, a, r+,w+, a+,rb,wb,ab,w+b,wb+
If(file == NULL)
{
        Printf(“Could not open a file!”);
}
else
{
        Char arr[20];
        fscanf(file,”19s”,arr);
        Printf()
        fclose(file);
}
return 0;
```

```
}
```

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

Int main(int argc, char*argv[])
{
Tmpnam();
        FILE* tempFile = tmpfile();    //Create a temp file
        Fprintf(tempFile);
        Fclose(tempFile);

return 0;
}
```

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

Int main(void)
{
        Int i;
        FILE*outFile = fopen(“test.txt”, “w”);
        If(outFile)
        {
                Fprintf(“This is definitely in a file, right?”);
                Fflush(outFile);
                scanf(“%i”, &i);
                Fclose(outFile);
        }
return 0;
}
```

What we learnt:

Fopen,

Fclose,

Fprintf,

Fscanf,

Fflush,

tmpfile / tmpnam,

feof (f end of file),

getc,

- fgetc

putc,

-fputc

fread

fwrite

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

Int main(void)
{
        FILE* file;
        char c;

        file = fopen(“main.c”, “r”);
        if(file)
        {
                Int c;
                Do
                {
                        C = getc(file);
                        /**********
                        idiom for looping over characters in a file while ((c= getc(file))!=EOF)
                        ************/
                        If(c==EOF)                      //last character
                        {
                                Break;
                        }
                        Printf("%c", (char)c);
                }while (1);
                /**********************
                while(!feof(file))
                {
                        Fscanf(file, "&c", &c);
                        Printf("&c", c)
                }
                ************************/
                fclose(file);
        }
        Return 0;
}
```

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

Int main(int argc, char**argv)
{
        FILE* inFile;
        FILE* outFile;

        If (argc!=3)
        {
                Printf(“Wrong number of parameters!”);
        Exit(1);
        }

        inFile = fopen(argv[1], “r”);
        if(inFile)
        {
                outFile =fopen(argv[2], “w”);
                if(outFile)
                {
                        char c;
                        do
                        {
                                fread()
                                if(fread(fread(&c, sizeof©,1,inFile)==0)
                                {
                                        break;
                                }
                                fwrite(&c, sizeof©,1,outFile);
                                fclose(outFile);
                        }while(1)
                fclose(inFile);
        Return 0;
}
```

Enumerations

```
#include <stdio.h>

typedef enum GameStates GameStates_t;
Void print(enum GameStates g)
```

```
{
Switch (g)
{
        Case MENU:
                Printf(“MENU”);
                Break;
        Case GAMEPLAY:
                Printf(“GAMEPLAY”);
                Break;
        Case CREDITS:
                Printf(“CREDIT”);
                Break;
        Case WIN_SCREEN:
                Printf(“WIN_SCREEN”);
                Break;

}


Int main(void)
{
        const int MENU = 0;
        const int GAMEPLAY = 1;
        const int CREDITS = 2;


        int current_gamestate = MENU;
        current_gamestate = 3;


        enum GameStates
        {
                 MENU,
                GAMEPLAY,
                CREDITS = 30,
                WIN_SCREEN

        }current_gamestate;


        Enum GameStates current_gamestate = WIN_SCREEN;

        (void)current_gamestate;


        Printf("%lu",sizeof(current_gamestate));

        Printf("%d",current_gamestate);


        Current_gamestate = 3;          //Do not do it, even if it compiles


        return 0;

}
```

Structures

```
#include <stdio.h>


struct StudentRecord

{

        Int student_id;

        double gpa;                             //data members

};


Struct StudentRecord s1;


Struct st1

{

        char a;

        Unsigned char b;

        Int c;
};

Struct str2
{
        char a;
        unsigned char b;
        int c;
        char d;
};
```

```
Struct StructureExample
{
        int dataMember1;
        char dataMember2;
        double dataMember3[2];
        signed char dataMember4;
};

123456789012345678901234567890123456789012345678901234567890
1111
    2###
        3333333333333333
                        4########
```

```
#include <stdio.h>

struct student
{
        unsigned long int id;
        Float gpa;
}

typedef struct student student_t;
int main(void)
{
        Student_t s1 = {0, 0.0f};
        FILE* file;

        scanf("%d", &(s1.id));
        scanf("%f", &(s1.gpa));
        file= fopen("student.data","rb");
        if(file)
        {
                If(fread(&s1, sizeof(student_t), 1,file) > 0)
                {
                        fread()
                        printf("Student ID: %d\n, s1.id);
                        printf("Student GPA: %f/n", s1.gpa);
                        fclose(file);
        }
        printf("Student ID: %d\n", s1.id);
        printf("Student GPA: %d\n", s1.gpa);

        file = fopen("student.txt", "wb");
        if(file)
        {
                fwrite(&s1, sizeof(student_t), 1, file);
                fclose (file);
        }
        return 0;
```

}

Segment in a program

Header

Text = string literals

RODATA = string literals

DATA = static variables that are not initialised to 0.

BSS

~HEAP

Stack (call stack)

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

Int main(void)
{
        /* realloc (old, newsize) */

        Unsigned Int n;
        Int*arr_ptr;
        Printf("Enter a number: ");
        If(Scanf("%d", &n) ==1)
        {
                // Arr_ptr=(int*)malloc(sizeof(int)*n);
                Arr_ptr = (int*)calloc(n, sizeof(int));
                If(arr_ptr!= NULL)
                {
                        Unsigned Int I;
                        For(i=0; i<n, ++i)
```

```
                    {
                    Printf("%d. ", i+1);
                    Scanf("%d" , arr_ptr + i);
            }
            Printf("The numbers are: \n");

            For(i=0l i<n; ++i)
            {
            Printf("%d. %d", I +1, arr_ptr[i]);
                    Free(arr_ptr);
            }
    }
    Return 0;
}
```

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

Int main(void)
{
    Double *p1 = NULL;
    P1 = (double*)malloc(sizeof(double)*1024*1024*1024);
    If(p1)
    {
            Int x = 0;
            *p1[120000] = 2.0;

            Printf("First malloc did the job!");
            Scanf(%d", &x);
```

```
                *p1 = 1.0;
                {
                Double*temp = NULL;
                temp = (double*)malloc(sizeof(double)*2);
                if(temp)
                {
                        Memcpy(temp, p1, sizeof(double))
                        Free(p1);
                        P1 = temp;
                }
        }
                Free(p1);
}
                Return 0;
}
```

```
#ifndef_STRING_H_
%DEFINE _STRING_H_

struct String:
typedef struct String String_t;
typedef String* PString;
PString create_string(const char*);
Void append_string(PString, const char*);
Int compare_string(PString, const char*);
Void destroy_string(PString);

#endif



#include <stdio.h>
```

```
#include “string.h”

Int main(void)

{

            PString a;

            a = create_string(“ABC”);

            append_string(a, “DEF”);


            if(compare_string(a), “ABCDEF”))

            {

            Printf(“Equal. “);

            }

            Else
            {

            Printf(“different!”);

            }

            Destroy_string(a);


            Return 0;

}
```

```
#include <stddef.h>

#include “string.h”


Struct String

{

            Char* data;

            Size_t length;

};

Int strlen(const char* text)

{

            Int i=0;

            While(*text++!=’\0’)
```

```
        {
                ++I;
        }
        Return I;
}
PString create_string(const char*)
{
        int I;
        PString result = (PString)malloc(sizeof(String));
        result ->length = strlen(text);
        result ->data = (char*)malloc(sizeof(char)*result ->length +1));

        do
        {
                Result -> data[i] = text[i];
        }while(test[i++]!='\0');

}
Void append_string(PString, const char*)
{
        Char*temp =(char*)malloc(lhs->length+sizeof(char*(strleng(rhs)+1);
        If(temp)
        {
                Int I =0;
                while(lhs->data[i] !='\0')
                {
                        Temp[i] = lhs ->data[i];
                        ++I;
                }
                do
                {
                        temp[i] =*rhs;

                        i++;

                }while (*rhs++ != ‘\0’)

        }

        }

}

Int compare_string(PString lhs, const char*)

{

        Int I = 0;

        Do

        {

                If(lhs ->data[i])

                {

                        Return 0;

                }

        }while(rhs[i++] !=’\0’);

        Return 1;

}

Void destroy_string(PString s)

{

        If(s!=NULL)

        {

                free(s->data);

                free(s);

        }

}
```