20IT205/20CS243 – DATA STRUCTURES

Pre-requisites: What should you be familiar with?!?

- Structure of C
- Data types
- Operators
- Conditional statements
- Control statements
- Loops
- Arrays
- Functions
- pointers
- Structures

Key Terms to know:

- Byte – unit of digital information (1 byte= 8 bits)
- Computer memory – hardware part of the computer which stores the data, information, instruction
- Main memory- temporary memory (RAM)
- Secondary memory – permanent memory (ROM)
- Compiler – converts high level language to low level language.it takes entire programs for conversion.
- Byte code – instruction that only CPU can understand
- Low level language – language understood by machine(0& 1)
- High level language - understood by humans(C program)
- Interpreter – converts high level language to medium level language. It takes only one line in a program at a time.
- IDE – combines several tools to write and test software.

Why C?

- Supports Structured programming ->procedural language
    - Structure here denotes functions/modules
- Used to build system components ->system programming

  - Works closely with processor.
  - Performance critical applications are built using C
- Operating system, compilers, interpreters, databases use C as their backbone
- Other Languages with C style syntax is popular – C++,c#,Java,Rust
- Same program can run on different computers → C is "portable"
- High level programming libraries rely on C → eg:python

Practical applications:

- Windows kernel -> written in C,C++,C#
- Linux kernel -> written in C
- Mac OS X -> written in C,C++
- Android OS → uses C
- Oracle , MySQL databases→ built using C
- Compilers -→Cfront
- Interpreters→Cpython
- Libraries → Numpy(python)

How to Code in C?

STEP 1 :INSTALL IDE(INTEGRATED DEVELOPMENT ENVIRONMENT)

- Used here : codeblocks → C/C++
- Compiler used : MinGW
- Link to download : link

STEP 2: LEARN GENERAL C STRUCTURE

GENERAL C STRUCTURE

	STEP	EXPLANATION	CODING
1	Preprocessors	• To load the pre-defined functions-printf(),scanf() • Written in header file • Saved by using .h extension • #include<header.h> searches within "include" folder • #include " header.h" searches in all folders	#include<stdio.h>
2	main() function	• Denotes starting point of the C program • int main() – use return 0; • void main – use nothing	Int main()
3	{	• denotes the starting of main() scope	{
4	Memory allocation/variable declaration	• allocates memory for the variables with their datatypes • terminate using ;	int a,b,c,d;

5	initialization	• assigning value to variables • use “&” address oprator to store value in the address of the variable. • run time – when you assign value to variables by getting them from output screen(using scanf()) • compile time- when you directly give value to variables in C program • terminated by semicolon	scanf(“ %d”,&a);//run time scanf(“%d%d”,&a,&b); a=10; // compile time
6	task	• perform given task	d=a+b+c+d;
7	output	• show the result of the task • don’t use “&” address operator here in printf()	printf(“%d”,d);
8	}	• denotes the end of the scope	}
9	comment lines	• used for more understanding • this wont go for compilation process	// heyyyy!!! (single line comments) /* hey (multiline comments) hi hello*/

STEP 3: LEARN TOKENS

CATEGORY	EXPLANATION	TYPES
Tokens	every small meaningful element to compiler	• Keywords – int,switch,break . . . • constants – 10,20 • identifiers – name of the function, array, variables : Eg : use int a1; not int 1a; use int a_1; not int a 1; • Strings = " hey hi" • Special symbols 1. [] – used for array element reference 2. ()-used in functions 3. , - used to separate elements 4. ; - statement terminator 5. * - pointer variable 6. = assignment(from right to left) • Operators 1.arithmetic (+,-,*,/,%) 2.relational(>,>=,<,<=,==,!=) 3.logical(&&,||,!) 4.assignment(=,+=,-=,*=,/+,%=) 5.bitwise(&,|,<<,>>,~,^) 6.unary(-,++,__,!,&,sizeof()) 7.ternary(? :)

STEP 4: LEARN INPUT&OUTPUT

scanf(" format_specifier",(addressof)variable_name); //use &

example : scanf("%d",&a);

	data type	bytes	example	format specifier	& (yes/no)	syntax
primitive data type	int	4	1	%d	yes	scanf("%d",&a);
	float	8	1.900000	%f	yes	scanf("%f",&a);
	char	1	d	%c	yes	scanf("%c",&a);
	boolean	1	0,1	no specific format,can use %d	yes	scanf("%d",&a);
	double	16	67.9090909090909090	%lf	yes	scanf("%lf",&a);
derived data types	integer array	4*n	{1.2.3}	%d	yes	scanf("%d",&a[i]);//use for loop
	float array	8*n	{1.2,2.3,3.4}	%f	yes	scanf("%f",&a[i]);// use for loop
	char array/string	1*n	{"gh,"hj"}	%s	no	scanf("%s",a);//no for loop
user defined data type	structure&union	add the individual byte allocation for the data type	{name,age,cgpa}	any primitive/derived data type format specifier	yes , but for string variables "no"	use the above syntaxes depending upon the datatype of variable name

printf("format_specifier",variable_name); //don't use &

example:printf("%d",a);

	data type	bytes	example	format specifier	& (yes/no)	syntax
primitive data type	int	4	1	%d	no	printf("%d",a);
	float	8	1.900000	%f	no	printf("%f",a);
	char	1	d	%c	no	printf("%c",a);

	boolean	1	0,1	no specific format,can use %d	no	printf(“%d”,a);
	double	16	67.9090909090909090	%lf	no	printf(“%lf”,a);
derived data types	integer array	4*n	{1.2.3}	%d	no	printf(“%d”,a[i]);//use for loop
	float array	8*n	{1.2,2.3,3.4}	%f	no	printf(“%f”,a[i]);// use for loop
	char array/string	1*n	{“gh,”hj”}	%s	no	printf(“%s”,a);//no for loop
user defined data type	structure&union	add the individual byte allocation for the data type	{name,age,cgpa}	any primitive/derived data type format specifier	no	use the above syntaxes depending upon the datatype of variable name

STEP 5: LEARN SYNTAX

CATEGORY	CLASSIFICATION	SYNTAX	EXPLANATION(IF ANY)
DECISION MAKING STATEMENTS • true block-if block/else if block • false block – else block • true(condition)→boolean value:1 • false(condition)→boolean value:0	**if**	if(condition){ }	• if the condition is true, statement inside the if block (true block) is executed. • if the condition is false, statements after if block is executed
	if-else	if(condition){ } else{ }	• if the condition is true, statement inside the if block (true block) is executed. • if the condition is false, statements inside else block is executed
	if-else if ladder **1- if block,else block** **n no.of else if blocks**	if(condition){ } else if(condition){ }	• if the condition is true, statement inside the if block (true block) is executed. • if the condition is false, else if block conditions are checked.

		. . . else{ }	1. If it is true,statements insode those blocks are executed • If the conditions of both if and all else if blocks are false, statements inside else block is executed
	nested if	if(condition){ if(condition){ } }	-
	nested if-else	if(condition){ if(condition){ } else{ } } else{ }	-
	nested else-if	if(condition){ } else{ if(condition){ } else{ } }	-
	switch case	switch(choice){ case 1: statement;	• Multiway branching statement

		break; case 2: statement; break; case n: statement; break; default: }	• Choice given and the corresponding case number match is alone executed. 1. remaining cases are not checked 2. break at the end of the loop ,transfers the control to the end of switch } • if choice doesn't match case number,default is executed
LOOPING STATEMENTS • loops runs until condition becomes false/loop only can run as long as condition is true ✓ step 1: starting point of loop/initialization ✓ step 2: test condition ✓ step 3: statement inside loop ✓ step 4: inc/dec • if u start i=0→loop goes from 0 to n-1. So test condition→i<n • if u start i=1→loop goes from 1 to n. So test condition→i<=n	**for loop**	for(step 1;step2;step4){ //step3; } eg: for(int i=0;i<n;i++){ printf("%d",i); }	• entry controlled loop-since the condition checked first • for(; ;) →infinite loop
	while loop	step 1; while(step2){ //step 3; step 4; } eg: int i=1; while(i<=n){ printf("%d",i); i++; }	• entry controlled loop-since the condition checked first • while(1)→infinite loop

	do…while loop	step 1; do{ step 3; step 4; }while(step 2); eg: int i=1; do{ printf("%d",i); i++; }while(i<=n);	• exit controlled loop-since the condition checked last
Unconditional jump	**continue** • used in loops within if blocks	for(; ;) { for(; ;){ if(condition){ continue; } } }	• resets the program control to the beginning of the loop
	goto	• *forward jump* goto label: ….. ….. label: statements; • *backward jump*	• unconditional transfer of control from one part of the program to the label part

		label: statements; ….. ….. goto label:	
	break	for(; ;){ for(; ;){ if(condition){ break; } } }statements;//this will get executed	• can be used only within loops/switch case • it can be used with if block incase of if block written inside the loop • it exits from the current loop,transfers control to the statements below loop.
	return	calling_function(){ return value/expression; } main(){ data type catch_the_value; catch_the_value=called_function(); }	• it is the value/expression returned by the called function to the calling function

STEP 6:1 D -ARRAYS

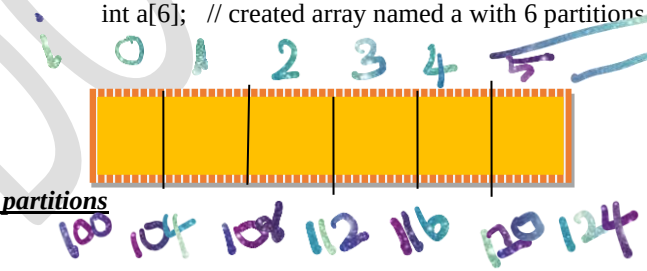
Step 1:

Declaration/memory allocation

Syntax: datatype array_name[size];

step 2 : initialization/assigning value to each 6 partitions

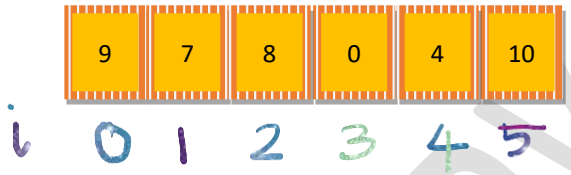
int a[6]; // created array named a with 6 partitions



type 1 : compile time initialization with size // giving value directly

in program

int a[6]={9,7,8,0,4,10}



type 2: compile time initialization without size

int a[]={9,7,8,0,4,10}

type 3: run time initialization // getting value in black output screen

- should use scanf()
- cant get input using only array name since we need 6 inputs for 6 partitions
- use index to refer each partition
- run for loop from 0 to n-1 index, use scanf to get value for each partition.
- syntax:
  for(int i=0;i<n;i++){
  scanf("%d",&a[i]);}
  - iteration 1 : i=0;0<6→True; get value through scanf() for &a[0] which is address 100.
  - iteration 2: i=1;1<6→True; get value through scanf() for &a[1] which is address 104.
  - iteration 3 : i=2;2<6→True; get value through scanf() for &a[2] which is address 108.
  - iteration 4 : i=3;3<6→True; get value through scanf() for &a[3] which is address 112.
  - iteration 5 : i=4;4<6→True; get value through scanf() for &a[4] which is address 116.
  - iteration 6 : i=5;5<6→True; get value through scanf() for &a[5] which is address 120.
  - iteration 7 : i=6;6<6→False; exits for loop

step 3: Task. Run for loop to visit values in each index to perform task given

eg: sum of elements of an array

code: for(int i=0;i<n;i++){ sum=sum+a[i];}

- set sum =0 since 0 is the identity operator for addition
- enter for loop::::iteration 1 : i=0; 0<6→True; Add 0+ a[0]→0+9 and store it in same sum variable as 9
- iteration 2: i=1;1<6→True; Add 9+ a[1]→9+7 and store it in same sum variable as 16

- iteration 3 : i=2;2<6→True; Add 16+ a[2]→16+8 and store it in same sum variable as 24
- iteration 4 : i=3;3<6→True; Add 24+ a[3]→24+0 and store it in same sum variable as 24
- iteration 5 : i=4;4<6→True; Add 24+ a[4]→24+4 and store it in same sum variable as 28
- iteration 6 : i=5;5<6→True; Add 28+ a[5]→28+10 and store it in same sum variable as 38
- iteration 7 : i=6;6<6→False; exits for loop

step 4: output. Run for loop If necessary to visit each index.

eg: sum of elements of an array

print sum

printf("sum of the array elements=%d",sum); // sum of the array elements =38

STEP 7:FUNCTIONS

- Block of statements that performs a specific task
- scope : function starts when a request is initiated & stops when a response is received.
- has two components
  - calling function (requesting function)
  - called function (responding function)

calling function	**called function**
✓ **calls the user defined function from main(),written inside main()** ✓ **syntax:** **catching variable = function name(formal arguments)**	✓ **user defined function written outside main()** ✓ **syntax:** **returntype function_name(actual arguments){** **// function task** **return value/variable/expression;** **}**
function name should be the same for both calling and called functions **argument data type should be the same, but variable name can differ.** **return type denotes the data type of value returned by the called function.**	

catching variable catches the value returned by called function.

calling function→calls called fun by passing arguments**→performs task inside called function--<returns the result to the calling function→**calling functions catches the value using a variable**→print the variable for result**

example :adding two numbers using function

```
#include<stdio.h>
int add(int a,int b){ // called function→responding function
int c;
c=a+b;
return c;           //return →response
}
int main(){
int a=5,b=10,d;
d=add(a,b);           //calling function →requesting function
printf("%d",&d);→output
}
```

BASED ON ARGUMENTS AND RETURN TYPE : 4 WAYS OF FUNCTIONS

WITH ARGUMENT,WITH RETURN TYPE

```
#include<stdio.h>
int add(int a,int b){ // called function→responding function
int c;
c=a+b;
return c;           //return →response
}
int main(){
int a=5,b=10,d;
```

WITH ARGUMENT,WITHOUT RETURN TYPE

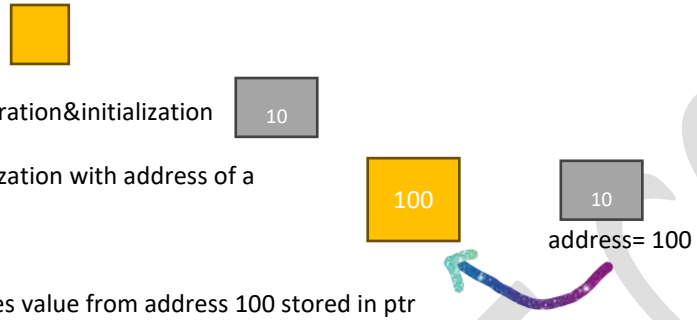
```
#include<stdio.h>
void add(int a,int b){ // called function→responding function
int c;
c=a+b;
printf("%d",c)           //response without return→print result here
}
int main(){
int a=5,b=10
```

d=add(a,b); //**calling function** →requesting function with catching variable **printf("%d",&d);**→output **}**	**//no need to declare d since we don't catch any value due to no return type** **add(a,b);** //**calling function** →requesting function with no catching variable since no need to return **}**
WITHOUT ARGUMENT WITH RETURN TYPE **#include<stdio.h>** **int add(){ // called function**→responding function **int a=5,b=10,c; // initialize a,b here since no arguments should be passed** **c=a+b;** **return c;** //**return** →response **}** **int main(){** **//no need to declare a,b since we don't pass arguments** **d=add();** //**calling function** →requesting function **printf("%d",&d);**→output **}**	**WITHOUT ARGUMENT WITHOUT RETURN TYPE** **#include<stdio.h>** **void add(){ // called function**→responding function **int a=5,b=10,c; // initialize a,b here since no arguments should be passed** **c=a+b;** **printf("%d",c);** //response→print the output here since no need to return **}** **int main(){** **//no need to declare a,b since we don't pass arguments** **//no need to declare d since we don't catch any value due to no return type** **add();** //**calling function** →requesting function **}**

STEP 8:POINTERS

- stores memory address of other variables,functions,pointers
- derived data type
- uses &-→referencing operator(address of a variable) , * dereferencing operator(value in the address)
- syntax : datatype *ptr;
  - ptr→pointer name
  - * in declaration is used to denote it's a pointer variable

```
#include<stdio.h>

int main(){

int *ptr; //pointer declaration

int a=10; //normal variable declaration&initialization

ptr=&a; //pointer variable initialization with address of a

printf("%d",a);     //10

printf("%d",*ptr);// 10 →retrieves value from address 100 stored in ptr

printf("%d",ptr);//100 →address of a in ptr

}
```



Types:

- integer pointer →int *ptr;
- array pointer→int *ptr=&arrayname
- structure pointer→ struct structurename *ptr;
- double pointer→ int **ptr;
- function pointer→int (*ptr)(int,int,int);

STEP 9:STRUCTURE IN C

- user defined datatype
- combined several primitive and derived datatypes
- allocates separate memory for each members

- structure is a blueprint from which several variables can be created
- no of bytes = sum of individual byte allocation of each member used within the structure

syntax:

```
struct structure_name{
datatype  member 1; //datatypes can be different
datatype  member 2;
            .
            .
datatype  member n;
} variable 1, variable 2;
```

general blue print

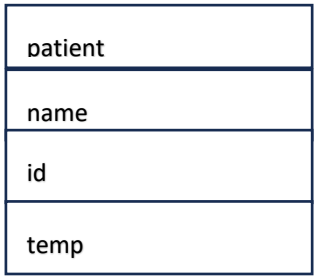
variables created for the structure

Example :

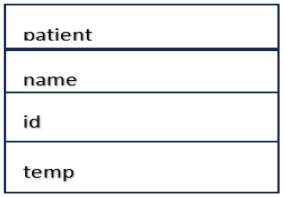
Patient structure

```
#include<stdio.h>
struct patient{
char name[10];
int id;
float temp;
} patient 1;
int main(){
scanf("%s%d%f",patient1.name,&patient1.id,&patient1.temp);
printf("%s%d%f",patient1.name,patient1.id,patient1.temp);}
```

blue print- patient



variable 1 ->patient 1



variable 2->patient