----------------------TEST YOUR "C" SKILLS-----------------------

CHAPTER **1**

**DECLARATIONS AND INITIALIZATION:-**

Que> O/P?

main()

{

char far \*s1,\*s2;

printf(""%d%d",sizeof(s1),sizeof(s2));

}

ans> 4 2

que>

o/p?

int x=40;

main()

{

int x=20;

printf("%d",x);

}

a>20

q>o/p?

main()

{

int x=40;

{

int x=20;

printf("%d",x0;

}

printf("%d",x);

}

a>20 40

q>is the following statement declaration or defination

extern int x;

a>declaration

q>

o/p?

main()

{

extern int i;

i=20;

printf("%d",sizeof(i));

}

a> error,i undefined

because extern int i is a declaration and not defination

q>is it true that the global variable have many declarations but only one defination?

a>yes

q>is it true that the function may have many decalaratins but only one defination?

a>yes

q>in the following program where the variable a is geting defined and where it is declared

main()

{

extern int a; /\* declaration\*/

printf("%d",a0;

}

int a=12; /\* defination\*/

q>wht will be the o/p of above program/

a>20

q>what is the difference between declaration and defination of a variable

a>declaration:-only gives the type,status and nature of variable without reserving any space for

the variable

defination;-actual space is reserverd for the variable and some initial value is given.

q>if the defination of the external variable occurs in the source file before it's use in a

perticular function then there is no need for an external declaration in the function

a>true

q>suppose the program is devided in three source files f1,f2,f3 and the variable is defined in file f1 but used in f2 and f3. In such a casewould we need the external declaration for

for the variable in files f2 and f3?

a>yes

q>when we mention the prototype of the function ,we are definig it or declaring it?

a>declaring it

q>what is the difference between following declarations

extern int fun()

int fun();

a>nothing except that that the firat one gives us hint that function fun is probally in another

file.

q>why does the following programreports the redeclaration error of function display()

main()

{

dispaly();

}

void dispaly()

{

printf("fggagaetaertrt");

}

a>here the function dispay() is called before it is declared .That is why the complier assumes it to be declared as

int display();

that accept unspecified no of arguments.i.e. undeclared function assumes to return int

on appering the declaration the fun shows that it returns void

hence the error

q>o/p?

main()

{

extern int fun(float);

int a;

a=fun(3.14);

printf("%d",a);

}

int fun(aa) /\* K & R style of function defenation\*/

float aa

{

return((int)aa);

}

a>error

because we have mixed the ansi prototype with k & r style of function defenation

If we use an ANSI prototype and pass float to the function then it is promoted to double

the funcction accepts it in to variable of type float hence the type mismatch occurs

To remady the situation define the function as

int fun(float aa)

{

.....

}

q>point error if any

struct emp

{

char name[20];

int age;

}

fun(int aa)

{

int bb;

bb=aa\*aa;

return(bb);

}

main()

{

int a;

a=fun(20);

printf("%d",a);

}

a>missing semicollon at the end of struct

due to which the function fun assumed to be returning vsr of type struct emp.

but it returns an int hence the error

q> If youare to share the variables or functions across several source files how would you enshore that all definications

and declarations are consistant?

a>The best arrangement is to place each defination in a revelent .c file , then put an external declaration in a header

file (.h file) and use #includeto briang the declaration wherever needed

The .c file which contains the definations should also include the header file, so that the complier can check that

the defination matches the declaration.

q>Correct the error

f(struct emp0;

struct emp

{

char name[20];

int age;

};

main()

{

struct empe={"Vivek",21}

f(e);

}

f(struct emp ee)

{

printf("\n %s %d",ee.name,ee.age);

}

a> declare the structure before the prototype of f.

q> Global variables are available to all functions. Does there exist a mechanism by way of which I can make it available

to some and not to others.

a>NO.

q>What do you mean by a translation unit

a>A trnslation unit is a set of source files as seen by the complier and translated as a unit. Generally one .c file

plus all header files mentioned in the #include directives

q>What wouldbe the output of the following program

main()

{

int a[5]={2,3}

printf("\n %d %d %d",a[2],a[3],a[4]);

}

a> 0 0 0

if a automatic array is partially initialised then remaiing elements are initialised by 0

q>o/p

main()

{

struct emp

{

char name[20];

int age;

float sal;

};

struct emp e={"vivek"}

printf("\n %d %f",e.age, e.sal);

}

a>0 0.000000

if an automatic structure is partially initialised then remaining elements are initialised bu 0.

q>Some books sugget that the fillowing definations should be preceded by the word static. Is it correct?

int a[]={2,3,4,12,32}

struct emp e={"vinod",23}

a>pre ANSI compilers has such requirment but compliers confirming to ANSI standard does not have such requirment.

q>point out error

main()

{

int(\*p)()=fun;

(\*p)();

}

fun()

{

printf("\n Loud and clear");

}

a>Here we are initialising function pointer to address of the function fun() but during the time of initialisation the function has not been defined. Hence an error

To eliminate the error add the prototype of function fun() before the declaration of p, as shown bellow;

extern int fun(); or simply

int fun();

q> point error if any

main()

{

union a

{

int i;

char ch[2];

};

union a z=512;

printf("%d %d",z.ch[0],z.ch[1]);

}

a>In pre-ANSI complier union vriable can not be initialised . ANSI complier permits initialisation of first member of the union

q>What do you mean by the scope of the variable? what are the 4 differet types of scopes that a variables can have?

a>Scope indicates the region over which the variable's declaration has an effect. The four kinds of scopes are: file

function,block,prototype.

q> what are different types of linkages?

a>There are three different types of linkages : external , internal , and none. External linkage means global, non-static

variables and functions, internal linkage means static variables and functions with file scope and no linkage means

local variables.

**CHAPTER 2**

**CONTROL INSTRUCTIONS.**

**1.**what would be the o/p of the following program

main()

{

int i=4;

switch(i)

{

default

printf(“\n a mouse”);

case 1:

printf(“\n a rabbit”);

break;

case 2:

printf(“\n a tiger’);

break;

case 3:

printf(“\n a lion”);

}

}

**ans**>a mouse

a rabbit

**2.** point out error in for loop if any-

main()

{

int i=0;

for( ; ; )

{

printf(“\n %d”,i++);

if(i>10)

break;

}

}

a.the condition in the for loop is must.

b.the two semicolons should be dropped

c.the for loop should replaced by a while loop

d.no error.

**Ans**:d

**3**. point error if any in the while loop

main()

{

int i=1;

while()

{

printf(“\n %d”, i++);

if(i>10)

break;

}

}

a.the condition in the while is must.

b.there should be at least one semicolon in the while()

c.the while loop must be replaced by a for loop.

d.no error.

**Ans**:a

**4**.point out error if any

main()

{

int x=1;

while(x<=5)

{  
 printf(“%d”,x);

if(x>2)

goto here;

}

}

fun()

{

here:

printf(“\n Nilesh”);

}

**ans**:goto can not take control to different function.

**5**.point error if any

main()

{

x=4,y=2;

switch(x)

{  
 case 1:

printf(“\n To error is human”);

break;

case y:

printf(“\n don’t do it here’);

break;

}

}

**ans**:constant expression required in second case we cant use y.

**6**.point error if any

main()

{

int x=1;

switch(x)

{  
 case 1:

printf(“\n Hellow”);

break;

case 1\*2+4:

printf(“\n the rock”);

break;

}

}

**ans**:no error constand expression like 1\*2+4 are acceptable in cases of switch.

**7.**point out error if any

main()

{

int a=1;

switch(a)

{  
 {  
 printf(“\n Programmers don’t die. They just lost in the procressing”);

}

**ans:** no error but switch with no case is not required.

**8**.point out error if any.

Main()

{

int x=1;

switch(x)

{  
 printf(“Hellow’);

case 1:

printf(“\n Nilesh”);

break;

case 2;

printf(“\n Vivek’);

break;

}

}

**ans**:though there is no error ,the first printf statement can never be executed irrespective of the value of x . In other words all the statements in the switch have to belong to some case or other.

**9**.Rewrite the following set of statements using conditional operator.

Int a=1,b;

If(a>10)

B=20;

**Ans**: int a,b,dummy;

a>10?b=20:dummy=1;

note that the following would not work

a>10?b=20: ; ;

**10**.point out error if any.

Main()

{

int a=10,b;

a>=5?b=100:b=200;

printf(“%d”,b);

}

**ans:** lvalue required in function main().The second assignment should be written in the paranthesis as follows

a>=5?b=100☹b=200);

11. O/p?

main()

{

char str[]=”part-time musicians are semiconductors”;

int a=5;

printf(a>10?”%50s”:”%s”,str);

}

1. part-time musicians are semiconductors
2. part-time musicians are semiconductors
3. error
4. none of above

ans: a

12. What is more efficient a switch statement or an if-else chain?

Ans: There is hardly any difference in efficiency in both cases. But one should use switch where it can be because it is a cleaner way to program.

13.Can we us switch statement to switch between strings.

Ans:No. cases in switch must be either integer constants or constant expressions.

14. We want to test whether the value lies between 2 to 4 or 5 to 7. can we do this using switch.?

Ans:Yes. But the way is not practical if the ranges are bigger. It is as shown bellow:-

switch(a)

{

case 2:

case 3:

case 4:

/\* some statements \*/

break;

case 5:

case 6:

case 7:

/\* some statements \*/

break;

}

15. The way break is used to take control out of switch can continue be used to take the control to the beginning of the switch.

Ans:No. continue can work only with loops and not with switch.

# CHAP 3

**EXPRESSIONS.**

1. O/p?

main()

{

static int a[20];

int x=0;

a[x]=x++;

printf(“\n %d %d %d”, a[0],a[1],x);

}

ans: 0 0 1

This is what some compliers give others may give a different answer. The reason is that the same statement causes the same object to be modified or to be modified and then inspected, the behavior is undefined.

1. O/p?

Main()

{

int x=3;

x=x++;

printf(“%d”,x);

}

ans: 4 but basically the behavior is undefined because of the same reason as above.

1. the expressions on the right hand side of the && and || operator does not get evaluated if the left hand side determines the output.

Ans: True. It is called short circuited mode of evaluating the logical expressions.

1. O/p?

Main()

{

int x=2;

printf(“%d %d”,++x,++x);

}

a 3 4

b 4 3

c 4 4

d o/p may vary from complier to complier

ans:d the order of evaluation of the arguments to a function call is unspecified.

1. O/p?

Main()

{

int x=10,y=20,z=5,a

a=x<y<z;

printf(“%d”,a);

}

a 1

b 0

c error

d none of above.

# Ans:a

6.Are the following statements same?

a<=20?b=30:c=30;

(a<=20)?b:c=30;

ans:No.

7.can you suggest the other way of writing the following expression such that 30 is used only once

a<=20?b=30:c=30;

ans: \*((a<=20)?&b:&c)=30;

8. How come the c standards says that the expression

x=y++ \* y++;

is undefined, whereas the expression

x=y++ && y++;

is perfectly legal.

Ans: According to C standards an object’s stored value can be modified only once (by evaluation of the expression ) between two sequence points. A sequence point occurs:

* At the end of full expression ( expression which is not a sub-expression in a larger expression).
* At the && , || , and ? : operator.
* At a function call (after the evaluation of all arguments, just before the actual call)

Since the first expression y is getting modified twice between two sequence points the expression is undefined. The second expression is legal because a sequence point is occurring at && and y is getting modified once after this sequence point.

9.If a[x]=x++ is undefined, then by the same reason x=x+1 should also be undefined. But it is not so. Why?

Ans:The standard says that if an object is to get modified within an expression then all accesses to it within the same expression must be for computing the value to be stored in the object. The expression a[x]=x++ is disallowed because one of the accesses of x (the one in a[x]) has nothing to do with the value that ends up being stored in x. In this case the complier may not know whether the access should take place before or after the incremented value is stored. Since there is no good way to define it , the standard declares it to be undefined. As against this the expression x=x+1 is allowed because x is accessed to determine x’s final value.

10.Would the expression \*p++ = c be disallowed by the complier.

Ans:No. Because even the value of p is accessed twice it is used to modify two different objects p and \*p.

11.In the following code in which order the functions would be called.

A=f1(23,65) \* f2(12/4) + f3();

1. f1,f2,f3.
2. f3,f2,f1.
3. The order may vary from complier to complier.
4. None of above.

Ans:C. Here the multiplication will happen before the addition, but in which order the function would be called is undefined.

12. In the following code in which order the functions would be called

a=(f1(23,65) \* f2(12/4) ) + f3();

a. f1,f2,f3.

b. f3,f2,f1.

c. The order may vary from complier to complier.

1. None of above.

Ans:C. Here the multiplication will happen before the addition, but in which order the function would be called is undefined. In an arithmetic expression the parentheses tell the complier which operands go with which operator but do not force the complier to evaluate everything within the parenthesis first.

13. What would be the output of the following program?

Main()

{

int x=-3,y=2,z=0,m;

m=++x && ++y || ++z;

printf(“\n %d %d %d %d”,x,y,z,m);

}

ans:-2 3 0 1

14 What would be the output of the following program?

Main()

{

int x=-3,y=2,z=0,m;

m=++y && ++x || ++z;

printf(“\n %d %d %d %d”,x,y,z,m);.

}

ans:-2 3 0 1

15. O/p?

main()

{

int x-3,y=2,z=0,m;

m=++x || ++y && ++z;

printf(“\n %d %d %d %d”,x,y,z,m)

}

ans: -2 2 0 1

16. O/p?

main()

{

int x-3,y=2,z=0,m;

m=++x && ++y && ++z;

printf(“\n %d %d %d %d”,x,y,z,m)

}

ans:-2 3 1 1

# Chap 4

**FLOATING POINT ISSUES**

1. O/P?

Main()

{

float a=0.7;

if(a<0.7)

printf(“C”);

else

printf(“C++”);

}

1. C
2. C++
3. Error
4. None of above.

Ans: A.

2.O/p?

main()

{

float a=0.7;

if(a<0.7f)

printf(“C”);

else

printf(“C++”);

}

a.C

b. C++

c. Error

1. None of above

Ans: B.

3.O/p?

main()

{

printf(“%f”,sqrt(36.0));

}

a.6.0

b.6

c.6.000000

d.Some absurd result

ans: D

4. Would this program give proper results <yes/no>?

main()

{

printf(“%f”,log(36.0));

}

ans:No. since math.h is not included.

5.Would the following printf()s print the same values for any value of a <yes/no>?

main()

{

float a;

scanf(“%f”,&a);

printf(“%f”,a+a+a);

printf(“%f”,3\*a);

}

ans: No.

6. We want to round off x, a float, to an int value. The correct way to do so would be

a. y=(int)(x+0.5);

1. y=int(x+0.5);
2. y=(int) x+0.5;
3. y=(int)((int)x+0.5);

ans: A.

7. Which error are you likely to get when you run the following program.

Main()

{

struct emp

{

char name[20];

float sal;

};

struct emp e[10];

int x;

for(x=0;x<9;x++)

scanf(“%s %f”, e[x].name,e[x].sal);

}

1. Suspicious pointer conversion.
2. Floating point formats not linked
3. Can not use scanf() for structure
4. Strings can not be nested inside structure

# Ans: B

8. What causes the error of problem 7 above to occur and how would you rectify the error in the above program.

Ans: What causes the “floating point format not linked” error to occur? When the complier encounters a referance to the address of the foloat , it sets a flag to have the linker link in the folating point emulator a floating point emulator is used to manipulate floating point numbers in runtime library functions like scanf() and atof(). There are some cases in which the reference to a float is a bit obscure and the complier does not detect the need for the emulator.

These situations usually occur during the initial stages of program development. Normally, once the program is fully developed, the emulator is used in such a fashion that the complier can accurately determine when to link in the emulator.

To force linking of the floating point emulator into an application just include the following function in your program

Void LinkFloat(void)

{

float a=0, **b=&a; /\*** cause the emulator to be linked. \*/

a=\*b; /\* suppress warning “var not used” \*/

}

There is no need to call this function from your program.

9. Which are the three different types of real data types available in C and what are the format specifiers used for them?

Ans:

Float 4 bytes %f

Double 8 bytes %lf

Long double 10 bytes %Lf

10.By default any real number is treated as

a float

b double

c long double

b depends on the memory model that you are using

ans: B.

11.What should you do to treat the constant 3.14 as a float?

# Ans: Use 3.14f

12. What should you do to treat the constant 3.14 as long double?

# Ans: Use 3.14l

13. O/p?

main()

{

printf(“%d %d %d”,sizeof(3.14f),sizeof(3.14),sizeof(3.14l));

}

1. 4 4 4
2. 4 garbage value garbage value
3. 4 8 10
4. error

ans: C

14.The binary equivalent of 5.375 is

a 101.101110111

b 101.011

c 101011

d none of above

ans: B.

15.How floats are stored in binary form?

Ans: Floating points numbers are represented in IEEE format. The IEEE format for floating point storage uses a sign bit, a mantissa and an exponent for representing the power of two(2).The sign bit denotes the sign of the number (0- positive) and (1- negative) the mantissa is represented in binary after converting it into its normalized form. The normalized form results in a mantissa whose most significant bit is always 1. The IEEE format takes the advantage of this by not storing this bit at all. The exponent is an integer stored in an unsigned binary format after adding a positive integer bias. This ensures that the stored exponent is always positive. The value of bias is 127 for floats and 1023 for doubles

16. A float occupies 4 bytes. If the hexadecimal equivalent of each of these bytes is A, B, C, and D, then when this float is stored in memory these bytes get stored in the order.

1. ABCD
2. DCBA
3. 0xABCD
4. 0xDCBA

ans: B.

17. If the binary equivalent of 5.375 in normalized form is 0100 0000 1010 1100 0000 0000 0000 0000, what would be the output of the following program?

Main()

{

float a=5.375;

char \*p;

int x;

p=(char\*)&a;

for(x=0;x<=3;x++)

printf(“%02x”,(unsigned char)p[x]);

}

a 40 AC 00 00

b 04 CA 00 00

c 00 00 AC 40

d 00 00CA 04

ans: C.

# Chap 5

**FUNCTIONS**

1.O/p?

main()

{

int a,b;

a=sumdig(123);

b=sumdig(123);

printf(“%d %d”,a,b);

}

sumdig(int n)

{

static int s=0;

int d;

if(n!=0)

{

d=n%10;

n=(n-d)/10;

s=s+d;

sumdig();

}

else

return(s);

}

ans: 6 12;

2.What error would the following function give on compilation.

F(int a,int b)

{

int a;

a=20;

return a;

}

a .missing parenthesis in return statement.

B .The function should be defined as int f(int a,int b)

C . Redeclaratin of a.

1. None of above.

Ans: C.

3.Thee is a mistake in the following code. Add a statement in it to remove it.

Main()

{

int a;

a=f(10,3.14);

printf(“%d”,a);

}

f(int aa,float bb)

{

return((float(aa)+bb);

}

ans: Add the following function prototype in main()

float f(int aa,float bb);

4. Point error in the following code.

Main()

{

int a=10;

void f();

a=f();

printf(“%d”,a);

}

void f()

{

printf(“HI”);

}

ans. In spite of decelerating that the function will return void the program is trying to collect the value in a variable.

5.Point error if any

main()

{

int b;

b=f(20);

printf(“%d”,b);

}

int f(int a)

{

a>20?return(10):return(20);

}

ans: Return statement can not be used in format as shown in the conditional operator instead it should be as follows

return(a>20?10:20);

6. A function can not be defined inside another function. <true/false>

ans: True.

7.Will the following function work?<yes/no>

f1(int a, int b)

{

return(f2(20));

}

f2(int a)

{

return(a\*a);

}

ans:YES.

8.What are following two notations of defining functions commonly known as

int f(int a, float b)

{

/\* some code \*/

}

int f(a,b)

int a, float b;

{

/\* some code \*/

}

ans: The first one is known as ANSI notation. And the second one is known as Kernighan and Ritche, or simply K & R notation

9.In function two return statements should not occuar. <True/False>

ans: FALSE.

10. In a function two return statements should not occur successively.<True/False>

ans: TRUE.

11.In C all functions except main() can be called recursively.

Ans:FALSE. Any function including main() can be called recursively.

12. Usually recursion works slower than loops. <True/False>

ans: TRUE.

13.Is it true that too many recursive calls may result in stack overflow?

Ans TRUE.

14. How many times the following program prints ‘Jambaree’?

main()

{

printf(“\n Jambaree”);

main();

}

1. infinite.
2. 32767 times
3. 65535 times
4. Till stack doesn’t overflow.

Ans: D

# CHAP 6

**THE C PREPROCESSOR**

1. If a file to be included doesn’t exist, the preprocessor flashes an error message.

Ans: TRUE.

1. The preprocessor can trap simple errors like missing declarations, nested comments or mismatch of braces.

Ans: FALSE.

1. O/p?

#define SQR(x) (x \* x)

main()

{

int a,b=3;

a=SQR(b+2);

printf(“\n%d”,a);

}

1. 25
2. 11
3. error
4. garbage value

ans:B. Because on preprocessing the expression becomes (3+2 \*3 +2).

4.How would you define SQR macro above in Q3 such as it gives the result of a as 25

ans: #SQR(x) ((x) \* (x))

1. O/p?

#define CUBE(x) (x \* x \* x)

main()

{

int a,b=3;

a=CUBE(b++);

printf(“\n %d %d”,a,b);

}

Ans: 27 6. Though some compilers may give this as an answer, according to ANSI C the expression b++ \* b++ \* b++ is undefined. Refer to chap 3 for more details.

6.Indicate what the SWAP macro be expanded to on preprocessing. Would the code compile?

#define SWAP(a,b,c) (int t; t=a, a=b, b=t;)

main()

{

int x=10,y=20;

SWAP(x,y,int);

Printf(“%d %d”,x,y);

}

Ans: (int t; t=a, a=b, b=t;);

This code won’t compile since the declaration of t can’t take place within parenthesis.

1. How should you modify SWAP macro such that it can swap two integers?

Ans: #define SWAP(a,b,c) c t; t=a, a=b, b=t;

8.What is the limitation of SWAP macro above In Q7?

Ans: It can’t swap pointer’s for example the following code won’t compile:-

#define SWAP(a,b,c) c t; t=a, a=b, b=t;

main()

{

float x=10,y=20;

float **p,**q;

p=&x; q=&y;

SWAP(p, q, float);

Printf(“%f %f”, x, y);

}

9.In which line of the following program, the error would be reported?

1. #define CIRCUM® (3.14 \* r\* r);
2. main()
3. {
4. float r=1.0.c;
5. c=CIRCUM(r);
6. printf(“\n %f”,c);
7. if(CIRCUM(r) == 6.28)
8. printf(“\n Good Day!”);
9. }

Ans: Line number 7. Whereas the culprit is really the semicolon in line 1. )n expansion line no 7 becomes if((3.14 \* 1.0 \*1.0);== 6.28). Hence the error.

10.What is the type of variable b in the following declaration?

#define FLOATPTR float \*

FLOATPTR a,b;

Ans: FLOAT. Since on expansion declaration becomes float \*a,b;

11.Is it necessary that the header files should have a .h extension?

Ans: NO.

12. What do the header files usually contain?

Ans: Preprocessor directives like #define, structure,union, and enum declarations, typedef declarations, global variable declarations and external function declarations. You should not write the actual code (function bodies) or global variable definition (that is defining or initializing instances) in header files. The #include directive should be used to pull in header files, not other .c files.

13.Would it result into an error if a header file is included twice?

Ans: YES. Unless the header file has taken care to ensure that if already included it doesn’t get included again.

14.How can a header file ensure that it doesn’t get included more than once?

Ans: All declarations must be written in a manner shown bellow. Assume that the name of the header file is FUNCS.H.

/\* FUNCS.H \*/

#ifndef \_FUNCS

#define \_FUNCS

/\* all declarations would go here \*/

#endif

Now if we include this file twice as shown bellow, it would get included only once

15.On inclusion, where are the header files searched for?

Ans: If included using < > the files get searched in the predefined (can be changed) include path. If included using “ “ syntax in addition to the predefined include path the files also get searched in the current directory (usually the directory from which you invoke the comlier).

16.Would the following #typedef work?

typedef #include l;

Ans: NO. Because typedef goes to work after preprocessing.

17. Would the following code compile correctily?

main()

{

#ifdef NOTE

/\* unterminated comment

int a;

a=10;

#else

int a;

a=20;

#endif

printf(“%d”,a);

}

Ans: NO. Even though #ifdef fails in this case (NOTE being undefined) and the if block doesn’t go for compilation errors in it are not permitted.

18. O/p?

#define MESS junk

main()

{

printf(“MESS”);

}

# Ans: MESS

19. Will the following program print the message infinite times? <Yes/No>

#define INFINITELOOP while(1)

main()

{

INFINITELOOP

Printf(“\n HELLOW”);

}

Ans

Ans: :YES.

20.O/p?

#define MAX(a,b) (a>b?a:b)

main()

{

int x;

x=MAX(3+2,2+7);

printf(“%d”,x);

}

# Ans: 9

21. O/p?

#define PRINT(int) printf(“%d”,int)

main()

{

int x=2,y=3,z=4;

PRINT(x);

PRINT(y);

PRINT(z);

}

# Ans: 2 3 4

22.O/P?

#define PRINT(int) printf(“int=%d”,int)

main()

{

int x=2,y=3,z=4;

PRINT(x);

PRINT(y);

PRINT(z);

}

# Ans: int=2 int=3 int=4

23.How would modify the macro of Q22 above such that it outputs

x=2 y=3 z=4

Ans:

#define PRINT(int) printf(#int”=%d”,int)

main()

{

int x=2, y=3, z=4;

PRINT(x);

PRINT(y);

PRINT(z);

}

The rule is if the parameter name is preceded by a # in the macro expansion, the combination (of # and parameter) will be expanded into a quoted string with the parameter replaced by the actual argument. This can be combined with string concatenation to print the output desired in our program. On expansion the macro becomes

Printf(“x” “=%d”,x);

The two strings get concatenated, so the effect is

Printf(“x=%d”,x);

24.Would the following program compile successively? <Yes/No>

main()

{

printf(“Tips” ”Traps”);

}

Ans: Yes. The o/p is TipsTraps.

25. Define the macro DEBUG such that the following program outputs:

DEBUG: x=4;

DEBUG: y=3.140000

DEBUG: ch=A

main()

{

int x=4;

float y=3.14;

char ch=’A’;

DEBUG (x,”%d”);

DEBUG (a,”%f”);

DEBUG (ch,”%c”);

}

Ans: #define DEBUG(var, Fmt) printf(“DEBUG:” #var ”=” #fmt ”\n”,ar)

26.O/p?

#define str(x) #x

#define Xstr(x) str(x)

#define oper multiply

main()

{

char \*opername=Xstr(oper);

printf(“%s”,opername);

}

# Ans: multiply

Here the two operations are being carried out expansion and stringizing. Xstr() macro expands its argument, and then str() stringizes it.

27. Write the macro PRINT for the following program such that it outputs:

x=4 y=4 z=5

a=1 b=2 c=3

main()

{

int x=4, y=4, z=5;

int a=1, b=2, c=3;

PRINT(x,y,z);

PRINT(a,b,c);

}

Ans: #define PRINT(var1, var2, var3) printf(“\n” #var1 “=%d”#var2 “=%d” #var3 “=%d”, var1,var2,var3)

# CHAPTER 7

**POINTERS**

1.Can you combine the following two statements into one

char \*p;

p=malloc(100);

Ans: char \*p=malloc(100);

2.Can you split the following statement into two statements?

Char far \*scr=(char far\*) 0xb8000000L;

Ans:

Char far \*scr;

Scr=(char far\*) 0xb8000000L;

3.Are the expressions \*ptr++ **and ++\***ptr same?

Ans: No. \***ptr++ increments the pointer and not the value pointed by it, whereas ++\***ptr increments the value being pointed to by ptr.

4. Can You write another expression which does the same job as ++\*ptr?

# Ans: (\*ptr)++

5.What would be the equivalent pointer expression for referring the same element as a[i][j][k][l]?

Ans: \*(\*(\*(\*(a+i)+j)+k)+l)

6.O/p?

main()

{

int arr[]=(12,13,14,15,16};

printf(“\n%d %d %d”,sizeof(arr),sizeof(\*arr),sizeof(arr[0]));

}

# Ans: 10 2 2

7.What would be the O/P of the program assuming that the array begins at 1002?

Main()

{

int a[3][4]={

1,2,3,4,

5,6,7,8,

9,10,11,12

};

printf(“\n%u %u %u”,a[0]+1,\*(a[0]+1),\*(\*(a+0)+1));

}

# Ans: 1004 2 2

8.What would be the output of the program assuming that the array begins at location 1002?

Main()

{

int a[2][3][4]={

{

1,2,3,4,

5,6,7,8,

9,1,1,2

},

{

2,1,4,7,

6,7,8,9,

0,0,0,0

}

}

printf(“\n %u %u %u %d”,a,\*a,\*\*a,\*\*\*a);

# Ans:1002 1002 1002 1

9.In the following program how would you print 50 using p?

main()

{

int a[]={10,20,30,40,50}

char \*p;

p=(char\*)a;

}

Ans: printf(“\n %d”,\*((int\*)p+4));

10.Where can one think of using pointers?

Ans: At lot of places,for eg

Accessing array or string elements

Dynamic memory allocation

Call by referance

Implementing linked lists, trees, graphs, and many other data structures

Etc.

11.In the following program add a statement in the function fun() such that the address of a gets stored in j.

main()

{

int \*j;

void fun(int\*\*);

fun(&j);

}

void fun(int\*\*k)

{

int a=10;

/\* add statement here\*/

}

Ans: \*k=&a;

12.How would you implement an array of three function pointers where each function receives two ints and return a float?

Ans: float (\*arr[3])(int,int);

13. Would the following program give a compilation error or warning?

main()

{

float i=10,\*j;

void \*k;

k=&i;

j=k;

printf(“\n%f”,\*j);

}

Ans: No. Hee no type casting is required while assigning the value to and from k because conversions are applied automatically when other pointer types are assigned to and from void\*.

14. Would the following program compile?

main()

{

int a=10,\*j;

void \*k;

j=k=&a;

j++;

k++;

printf(“\n %u %u”,j,k);

}

Ans: No. An error would be reported in the statement k++ since arithmetic on void pointers is not permitted unless the void pointer is appropriately typecasted.

15. Would the following code compile successfully?

main()

{

printf(“%c”,7[“Sundaram”]);

}

Ans: YES. Prints m of sundaram.

# CHAPTER 8

**MORE ABOUT POINTERS**

1. Is the NULL pointer same as an uninitialised pointer?

Ans: NO.

1. In which header file is the NULL macro defined?

# Ans: In files stdio.h & stddef.h

1. Why is it that for a large memory model NULL has been defined as 0L and for small memory model as just 0?

Ans: Because in small memory model the pointer is two bytes long whereas in large memory models it is 4 bytes long.

1. What is a null pointer?

Ans: For each pointer type (like sat a char pointer) C defines a special pointer value which is guaranteed not to point to any object or function of that type. Usually the null pointer constant used for representin a null pointer is the integer 0 .

1. What’s the difference between a null pointer, a NULL macro, the ASCII NUL character and a null string?

Ans:

A null pointer is a pointer which doesn’t point anywhere.

A NULL macro is used to represent the null pointer in source code. It has the value 0 associated with it.

The ASCII NUL character has all it’s bits as 0 but doesn’t have any relationship with the null pointer.

The null string is just another name for an empty string “”.

6.O/p?

#include “stdio.h”

main()

{

int a,b=5;

a=b+NULL;

printf(“%d”,a);

}

# Ans: 5

1. Is the programming style adopted in Q6 good?

Ans: NO.

Only in context of pointers should NULL and 0 be considered equivalent. NULL should not be used when other kind of 0 is required. Even though this may work it is a bad style of programming. ANSI C permits definition of NULL macro as ((void\*)0), which ensures that the NULL will not work in non pointer contexts.

1. O/p?

#include “stdio.h”

main()

{

printf(“%d %d”,sizeof(NULL),sizeof(“”));

}

# Ans: 2 1

9.How many bytes are occupied by near, far, and huge pointers?

Ans: A near pointer is 2bytes long. The far and huge pointers are 4 bytes long.

10.What does the error “Null Pointer Assignment” mean and what causes this error?

Ans: The Null Pointer Assignment error is generated only in small and medium memory models. This error occurs in program which attempt to change the bottom of the data segment

In Borland’s C or C++ compilers, Borland places four zero bytes at the bottom of the data segment, followed by the Borland copyright notice “Borland C++- Copyright 1991 Borland Intl.”. In the small and medium memory models, a null pointer points to DS:0000. Thus assigning value to the memory referenced by this pointer will overwrite first zero byte in the data segment. At program termination, the four zeros and the copyright banner are checked. If either has been modified, then the Null Pointer Assignment error is generated. Note that the pointer may not truly be null, but may be a wild pointer that references these key areas in the data segment.

11.How do we debug a Null Pointer Assignment error?

Ans: In the IDE set two watches on the key memory locations modified in Q10 above. These watches, and what they should display in the watch window, are:

**(char\***)4,42MS Borland C++- Copyright 1991 Borland Intl.”

(char\*)0 00 00 00 00

Of course,the copyright banner will vary depending on your version of Borland C/C++ compiler.

Step through your program using F8 or F7 and monitor these values in the watch window. At the point where one of them changes, You have just executed a statement that uses a pointer that has not been properly initialized.

The most common cause of this error is probably declaring a pointer and then using it before allocating memory for it. For example, compile the following program in small memory model and execute it:

#include”dos.h”

#include”stdio.h”

#include”string.h”

main()

{

char \***ptr,\***banner;

banner=(char\*)MK\_FP(\_DS,4);

printf(“banner:%s\n”,bnner);

strcpy(ptr,”The world cup saga”);

printf(“”&ptr=%fp\n”,(void far\*)&ptr[0]);

printf(“banner:%s\n”,banner);

}

One of the best debugging techniques for catching Null Pointer Assignment error is to turn all warning compiler messages. If the above program compiled with warning turned off, no warning messages will be generated. However if all warning are turned on, both the strcpy() and printf() calls using the pointer variable will generate warnings. You should be particularly suspicious of any warning that a variable might be used before being initialized, or of a suspicious pointer assignment.

Note that the Null Pointer Assignment error is not generated in all models. In the compact, large and huge memory models, far pointer are used for data. Therefore a null pointer will reference 0000:0000, or the base of system memory, and using it will not cause a corruption of the key values at the base of data segment . Modifying the base of system memory usually causes system crash. Although it would be possible that a wild pointer would overwrite a key values, it would not indicate a null pointer. In the tiny memory model, DS=CS=SS. Therefore, using a null pointer will overwrite the beginning of the code segment.

12.Can anything generate a Null Pointer Assignment error?

Ans:YES.

A wild pointer that happens to reference the base area of the data segment may cause the same error since this would change the zeros or the copyright banner. Since data corruption or stack corruption could cause an otherwise valid pointer to be corrupted and point to the base of the data segment, any memory corruption could result in this error being generated. If the pointer used in the program statement which corrupts the key values appears to have been properly initialized, place a watch on that pointer. Step through program again and watch for it’s value (address) to change.

13. Are the three declarations char \*\*apple, char \*orange[] and char cherry[][] same?

Ans:NO.

14.Can two different near pointer contain two different addresses but refer to the same location in the memory?

Ans: NO.

15: Can two different far pointers contain different addresses but refer to same location in the memory?

Ans: YES.

16. Can two different huge pointers contain two different addresses refer to the same location in memory?

Ans: NO.

17. Would the following program give any warning on compilation?

#include “stdio.h”

main()

{

int \*p1,i=25;

void \*p2;

p1=&i;

p2=&i;

p1=p2;

p2=p1;

}

Ans: NO.

18. Would the following program give any warning on compilation?

#include “stdio.h”

main()

{

float \*p1,i=25.50;

char \*p2;

p1=&i;

p2=&i;

}

Ans: YES. Suspicious pointer conversion in function main.

19. What warning would be generated on compilation of the following program?

main()

{

char far \*scr;

scr = 0xb8000000;

\*scr=’A’;

}

Ans: Non-portable pointer assignment in function main.

20.How would you eliminate the warning generated on compiling the following program?

main()

{

char far \*scr;

scr = 0xb8000000;

\*scr=’A’;

}

Ans: Use the typecast scr =(char far \*)0xb8000000;

21. How would obtain a far address from the segment and offset addresses of the memory location?

Ans:

#include “dos.h”

main()

{

char \*seg = (char \*) 0xb000;

char \*off = (char \*) 0x8000;

char far \*p;

p=MK\_FP(seg,off);

}

22. How would you obtain segment and offect address from a far address of a memory location?

Ans:

#include “dos.h”

main()

{

char far \*scr = 0xb8000000;

char \*seg, \*off;

seg= (char \*) FP\_SEG(scr);

off = (char \*) FP\_OFF(scr);

}

23. In the large data model (compact, large, huge) all pointers to data are 32 bit long, whereas in a small data model (tiny, small, medium) all pointers are 16 bit long.

Ans: TRUE.

24. A near pointer uses the contents of CS register (if the pointer is pointing to code) or contents of DS register (if the pointer is pointing to data) for the segment part, whereas the offset part is stored in the 16-bit near pointer.

Ans: TRUE.

25. O/p?

main()

{

char far \*a=0x00000120, \*b=0x00100020, \*c= 0x00120000;

if(a==b)

printf(“\nHELLO”);

if(a==c)

printf(“\n HI”);

if(b==c)

printf(“\n HI HELLO”);

if(a>b && a>c && b>c)

printf(“\n BYE”);

}

Ans: BYE.

Here a,b and c refer to same location in the memory still the first three ifs fail because while comparing the far pointers using == (and !=) the full 32 bit value is used and since 32 bit values are different the ifs fail. The last if however gets satisfied, because while comparing using > (and >=, <, <=) onlu the offset value is used for comparison. And the offset values of a, b and c are such that the last condition is satisfied.

26. O/p?

main()

{

char huge \*a=0x00000120, \*b=0x00100020, \*c= 0x00120000;

if(a==b)

printf(“\nHELLO”);

if(a==c)

printf(“\n HI”);

if(b==c)

printf(“\n HI HELLO”);

if(a>b && a>c && b>c)

printf(“\n BYE”);

}

Ans: HELLO

HI

HI HELLO

Unlike far pointers, huge pointers are ‘normalized’ to avoid the strange behaviour as in Q25 above. A normalized pointer is a 32 bit pointer which has a much of its value in the segment address as possible. Since the segment can start every 16 bytes, this means that the offset will only have the value from 0 to F. Huge pointers are always kept normalized. As a result for any given memory address there is only one possible huge address-segment:offset pair for it

# CHAPTER 9

**ARRAYS**

1. O/p?

main()

{

char a[]=”Visual C++”

char \*b= “Visual C++”

printf(“\n %d %d”, sizeof(a), sizeof(b));

printf(“\n %d %d”, sizeof(\*a), sizeof(\*b));

}

Ans:

1. 2
2. 1
3. For the following statements would arr[3] and ptr[3] fetch the same character?

char arr[]=”Surprised”

char \*ptr=”Surprised”

Ans:YES.

1. For the statements in Q2 does the compiler fetch the character arr[3] and ptr[3] in the same manner?

Ans: NO.

For arr[3] the compiler generates code to start at location arr, move three past it, and fetch the character there. When it sees the expression ptr[3] it generates to start at location stored in ptr, add three to the pointer, and and finally fetch the character pointed to.

In other words, arr[3] is three places past the start of the object named arr, whereas ptr[3] is three places past the object pointed to by ptr.

1. What would be the o/p of the following program, if the array begins at address 1200?

main()

{

int arr[]={2, 3, 4, 1, 6};

printf(“%d %d”, arr, sizeof(arr));

}

# Ans: 1200 10

1. Does mentioning the array name gives the base address in all the contexts?

Ans: NO.

Whenever mentioning the array name gives its base address it is said that the array has decayed in to a pointer. This decaying does take place in two situations:

* + When array name is used with sizeof operator
  + When the array name is an operand of the & operator

1. What would be the O/p of the following program, if the array begind at address 65486?

main()

{

int arr[]={12, 14, 15, 23, 45};

printf(“%u %u”,arr, &arr);

}

# Ans: 65486 65486

1. Are the expressions arr and &arr same for an array of 10 integers?

Ans: NO.

Even though both may give the same address as in Q6 above they mean two different things. arr gives the address of the first integer, whereas &arr gives the address of array of ints. Since these address happen to be same the result of the expressions are same.

1. What be the output of the following program if the array begins at 65486?

main()

{

int arr[]= {12, 14, 15, 23, 45}

printf(“%u %u”, arr+1, &arr+1);

}

# Ans: 65488 65496

1. Wher are char a[] and char \*a treated as same by the compiler?

Ans: When using them as formal parameters while defining a function.

1. Would the following program compile successfully?

main()

{

char a[]= “Sunstroke”;

char \*p= “Coldwave”;

a=”Coldwave”;

p=”Sunstroke”;

printf(“\n %s %s”, a, p);

}

Ans: NO. Because we may assign a new string to a pointer but not to an array.

1. O/p?

main()

{

float a[]= { 12.4, 2.3, 4.5, 6.7};

printf(“\n %d”, sizeof(a)/sizeof(a[0]));

}

# Ans: 4

1. A pointer to a block of memory is effectively same as an array.

Ans: TRUE.

1. What would be the o/p of the following program if the array begins at 65472?

main()

{

int a[3][4] = {

1, 2, 3, 4,

4, 3, 2, 1,

7, 8, 9, 0

};

printf(“\n %u %u”, a+1, %a+1);

}

# Ans: 65480 65496

1. What does the following declaration mean

int (\*ptr)[10];

Ans: ptr is a pointer to an array of 10 integers.

1. If we pass the mane of a 1-D array to a function it decays into a pointer to an int. If we pass the name of 2-D array of integers to a function what would it decay into?

Ans: It decays into a pointer to an array and not a pointer to a pointer.

1. How would define the function f() in the following program?

int arr[MAXROW][MAXCOL};

fun(arr);

Ans:

fun ( int a[][MAXCOL])

{

}

or

fun (int (\*ptr)[MAXCOL]) /\* ptr is pointer to an array\*/

{

}

1. O/p?

main()

{

int a[3][4] = {

1, 2, 3, 4,

4, 3, 2, 8,

7, 8, 9, 0

};

int \*ptr;

ptr = &a[0][0];

fun (&ptr);

}

fun (int \*\*p)

{

printf(“\n%d”,\*\*p);

}

Ans: 1

# CHAPTER 10

**STRINGS**

1. O/p?

main()

{

printf(5+ “Fascimile”);

}

1. Error
2. Fascimile
3. mile
4. None of above

# Ans: C

1. O/p?

main()

{

char str1[]= “Hello”;

char str2[]= “Hello”;

if(str1==str2)

printf(“\n Equal”);

else

printf(“\nUnequal”);

}

a.Equal

b.Unequal

c. Error

1. None of above

# Ans: B

1. O/p?

main()

{

printf(“%c”,”abcdefgh”[4]);

}

1. Error
2. d
3. e
4. abcdefgh

# Ans: C

1. O/p?

main()

{

char str[7]= “Strings”;

printf(“%s”,str);

}

1. Error
2. Strings
3. Can not predict
4. None of above

Ans: C

Here str[] has been declared as a 7 character array into it a 8 character string has been stored. This would result into overwriting of the byte beyond the seventh byte reserved for the array with a ‘\0’. There is always a possibility that something important gets overwritten which would be unsafe.

1. How would you output \n on screen?

Ans: printf([\\n](file:///\\n));

1. O/p?

main()

{

char ch=’A’;

printef(“%d %d”, sizeof(ch), sizeof(‘A’));

}

1. 1 1
2. 1 2
3. 2 2
4. 2 1

Ans: B.

1. O/p?

main()

{

printf(“\n %d %d %d”, sizeof(‘3’), sizeof(“3”), sizeof(3));

}

1. 1 1 1
2. 2 2 2
3. 1 2 2
4. 1 1 1

Ans: B.

1. Is the following program correct?

main()

{

char \*str1= “United”;

char \*str2= “Front”;

char \*str3;

str3=strcat(str2, str2);

printf(“\n %s”,str3);

}

Ans:NO.

Since what is present in the memory bruond ‘United’ is not known and we are attaching ‘Front at the end of it, thereby overwriting something.

1. How would you improve the code in Q8 above?

Ans:

main()

{

char str1[15]= “United”;

char \*str2= “Front”;

char \*str3;

str3= strcat(str1, str2);

printf(“\n %s”,str3);

}

1. In the following code which function would get called, the user defined strcpy() or the one in the standard library?

main()

{

char str1[]= “Keep India Beautiful… emigrate!”;

char str2[40];

strcpy(str2, str1);

printf(“\n %s”, str2);

}

strcpy(char \*t, char \*s)

{

while(\*s)

{

\*t = \*s;

t++;

s++;

}

\*t= ‘\0’;

}

Ans: User defined strcpy().

1. Can you compact the code in strcpy() into one line?

Ans:

strcpy( char \*t, char \*s)

{

while (\*t++ = \*s++);

}

1. O/p?

main()

{

char \*str[] = { “Frogs”, “Do”, “Not”, “They”, “Croak!”};

printf(“%d %d”, sizeof(str), sizeof(str[0]));

}

# Ans: 12 2

1. How would you find the length of each string in the program above?

Ans:

main()

{

char \*str[] = {“Frogs”, “Do”, “Not”, “They”, “Croak!”};

int i;

for(i=0; i<=4; i++)

printf(“\n %s %d”, str[i], strlen(str[i]));

}

1. What is the difference in the following declarations

char \*p = “Samuel”;

char a[] = “Samuel”;

Ans: Here a is an array big enough to hold the message and the ‘\0’ following the message. Individual characters within the array can be changed but the address of the array would remain same.

On the other hand p is a pointer, initialized to point to a string constant. The pointer p may be modified to point to another string, but if you attempt to modify the string at which p is pointing the result is undefined.

1. While handling the string do we always have to process it character by character or there exists a method to process the entire string as one unit.

Ans: A string can be processed only on a character by character basis.

# CHAPTER 11

**STRUCTURES, UNIONS, AND ENUMERATIONS**

1. what Is the similarity between a structure, union and an enumeration?

Ans: All of them let you define new data type.

1. Would the following declaration work

typedef struct s

{

int a;

float b;

}s;

Ans: YES.

1. can a structure contain a pointer to itself?

Ans: Certainly, such structures are called self-referential structure.

1. point out error if any in the following code

typedef struct

{

int data;

NODEPTR link;

}\*NODEPTR;

Ans: A typedef defines a new name for a type, and in similar cases like the one shown bellow you can define a new structure type and a typedef for it at the same time.

typedef struc

{

char name[20];

int age;

}emp;

However, in the structure defined in Q4 there is an error because a typedef declaration can not be used until it is defined. In the given code fragment the typedef declaration is not yet defined at the point where the link field is declared.

1. How will you eliminate the problem in Q4 above?

Ans: To fix this code, first give the structure name (“struct node”). Then declare the link field as a simple struct node \* as shown bellow:

typedef struct node

{

int data;

struct node \*link;

}\*NODEPTR;

Another way to eliminate the problem is to disentangle the typedef declaration from the structure definition as shown bellow:

struct node

{

int data;

struct node \*link;

};

typedef struct node \*NODEPTR;

Yet another way to eliminate the problem is to precede the struct declaration with the typedef, in which case you could use the NODEPTR typedef when declaring the link field as bellow:

typedef structure node \*NODEPTR;

struct node

{

int data;

NODEPTR next;

};

In this case, you declare a new typedef name involving struct node even though struct node has not been completely defined yet; this you’re allowed to do. It is a matter of style which of the above solutions would prefer.

1. Point out error if any in the following code.

void modify(struct emp \*)

struct emp

{

char name[20];

int age;

};

main()

{

struct emp e = {“sanjay”,35};

modify(&e);

printf(“\n %s %d”, e.name, e.age);

}

void modify(struct emp \*p)

{

strupr(p->name);

p->age=p->age+2;

}

Ans: structure emp is mentioned in the prototype of the function modify() before defining the structure. To solve the problem just put the prototype after declaration of the structure or the just add the statement struct emp before the prototype.

1. Would the following code work?

#include<alloc.h>

struct emp

{

int len;

char name[1];

};

main()

{

char newname[]= “Rahul”;

struct emp **p= (struct emp\***) malloc (sizeof(struct emp)-1+ strlen(newname)+1);

p->len=strlen(newname);

strcpy(p->name,newname);

printf(“\n %d %s”, p->len, p->name);

}

Ans: YES, the program allocates space for the structure with the size adjusted so that the name field can hold the requested name (not just one character declaration would suggest). I don’t know whether it is legal or portable. However, the code did work on all the compilers that I have tried it with.

1. can you suggest a better way to write the program in Q7 above?

Ans:- The truly safe way to implement the program is to use a char pointer instead of an array

#include<alloc.h>

struct emp

{

int len;

char \*name;

};

main()

{

char newname[]=”Rahul”;

struct emp \*p=(struct emp\*) malloc(sizeof(struct emp));

p->len=strlen(newname);

p->name=malloc(p->len+1);

strcpy(p->name,newname);

printf(“\n%d %s”,p->len,p->name);

}

Obviously, the convenience of having the length and the string stored in the same block of memory has now been lost, and freeing instances of this structure will require two calls to the function free();

1. How would you free the memory allocated in Q8 above?

Ans:- free(p->name);

free(p);

1. Can you rewrite the program in Q8 such that while freeing the memory only one call to free() would suffice?

Ans:-

#include<alloc.h>

struct emp

{

int len;

char \*name;

}

main()

{

char newname:- “Rahul”;

char \*buf=malloc(sizeof(strut emp)+strlen(newname)+1);

struct emp \*p= (struct emp \*) buf;

p->len=strlen(newname();

p->name=buf+sizeof(struct emp);

strcpy(p->name,newname);

printf(“\n %d %s”,p->len,p->name);

free(p);

}

1. O/p?

main()

{

struct emp

{

char \*n;

int age;

};

struct emp e1={“Dravid”,23};

struct emp e2=e1;

strupr(e2.n);

printf(“\n%s”,e1.n);

}

Ans:- DRAVID

When a structure is assigned, passed, or returned, the copying is done monolithically. This means that the copies of any pointer fields will point to the same place as the original. In other words, anything pointed to is not copied. Hence on changing the name through e2.n it automatically changed e1.n.

1. Point out error if any in the following code.

main()

{

struct emp

{

char n[20];

int age;

};

struct emp e1= {“Dravid”,23};

struct emp e2=e2;

if(e1==e2)

printf(“\n The structures are equal”);

}

Ans:- Structures can not be compared using the built in == and != operations. This is because there is no single, good way for a compiler to implement structure comparison. A simple bute by byte comparison could fail while comparing the bits present in unused padding in the structure (such padding is used to keep the alignment of letter fields correct). A field by field comparison might require unacceptable amounts of repetitive code for large structures. Also any compiler generated comparison would not be expected to compare pointer fields appropriately in all cases; for example, it is often appropriate to compare char\* fields with strcmp() rather than with ==;

1. How would you check whether the contents of two structure variable are same or not?

Ans:-

Struct emp

{

char n[20];

int age;

};

main()

{

struct emp e1={“Dravid”,23};

struct emp e2;

scanf(“%s %d”,e2.n,e2.age);

if(structcmp(e1,e2)==0)

printf(“\n Equal”);

else

printf(“\n Unequal”);

}

structcmp(struct emp x, struct emp y)

{

if(strcmp(x.n,y.n)==0)

if(x.age==y.age)

return(0);

return(1);

}

1. How are structure passing and returning implemented by the compiler?

Ans:- When structures are passed as arguments to functions, the entire structure is typically pushed on the stack. To avoid the overhead many programmers often prefer to pass pointers to structures instead of structures. Structures are often returned from functions in a location pointed by an extra, compiler supplied “hidden” argument to the function.

1. How can I read/write structures from/to data files?

Ans:- To write out a structure we can use fwrite() as shown bellow:

Fwrite(&e, sizeof(e), 1, fp);

Where e is the structure variable. A corresponding fread() invocation can read the structure back from the file.

On calling fwrite() it writes out sizeof(e) bytes from the address &e. Data files are written as memory images with fwrite(), however will not be portable, particularly if they contain floating point fields or pointers. This is because the memory layout of structures is machine and compiler dependent. Different compilers may use different amount of padding, and the size and byte orders of fundamental types vary across machines. Therefore structures written as memory images cannot necessarily be read back in by programs running on other machines (or even compiled by other compilers) and this is an important concern if the data files you’re writing will ever be interchanged between machines.

1. If the following structure is written to a file using fwrite(), can fread() read it back successfully?

Struct emp

{

char \*n;

int age;

};

struct emp e= {“Sujay”,15};

file \*fp;

fwrite(&e, sizeof(e), 1, fp);

Ans:- No, since the structure contains a char pointer while writing a structure to the disk using fwrite() only the value stored in the pointer n would get written (and not the string pointed by it). When this structure is read back the address will be read back but it is quite unlikely that the desired string would be present at this address in memory.

1. Would the following program always output the size of structure as 7 bytes?

Struct ex

{

char ch;

int i;

long int a;

};

Ans:- No. A compiler may leave holes in structures by padding the first char in the structure with another byte just to ensure that the integer that follow is stored at an even location. Also there might be two extra bytes after the integer to ensure that the long integer is stored at an address which is a multiple of 4. This is done because many machines access values in memory most efficiently when the values are appropriately aligned. Some machines can not perform unaligned accesses at all and require that all the data be appropriately aligned.

Your compiler may provide an extension to give you control over the packing of structures (i.e. whether they are padded), perhaps with a #pragma, but there is no standard method.

If you’re worried about wasted space, you can minimize the effects of padding by ordering the members of the structure from largest to smallest. You can sometimes get more control over size and alignment by using bitfields, although they have their own drawbacks.

1. What error does the following program give and what is the solution for it?

main()

{

struct emp

{

char name[20];

float sal;

};

struct emp e[10];

int i;

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

scanf(“%s %f”, e[i].name, &e[i].sal);

}

Ans:- Error :Floating point formats not linked. What causes this error to occur? When the compile encounters a reference to the address of the float, it sets a flag to have the linker link the floating point emulator.

A floating point emulator is used to manipulate floating point numbers in runtome library functions like scanf() and atof(). There are some cases in which the reference to a float is a bit obscure and the compiler does not detect the need for the emulator.

These situations usually occur during the initial stages of program development. Normally once the program is fully developed, the emulator will be used in such a fashion that the compiler can accurately determine when to link the emulator.

To force linking of the floating point emulator into an application, just include the following function in your program.

Void linkfloat(void)

{

float a=0, **b=&a; /** causes emulator to be linked\*/

a=\*b; /\*suppress warning –var not used\*/

}

There is no need to call this function in your program.

1. How can I determine the byte offset of a field within a structure?

Ans:- You can use the offset macro given bellow. How to use this macro has also been shown in the program.

#define offset(type, mem) ( (int) ( (char\*) & ( ( type\*) 0)- mem – (char\*) (type\*) 0 ) )

main()

{

struct a

{

char name[15];

int age;

float sal;

}

int offsetofname, offsetofage, offsetofsal;

offsetofname=offset(struct a, name);

printf(“\n%d”, offsetofname);

offsetofage=offset(struct a, age);

printf(“\n%d”,offsetofage);

offsetofsal=offset(struct a, sal);

printf(“\n%d”,offsetofsal);

}

The output of this program will be

0

15

17

1. The way mentioning the array name or function name without [ ] or ( ) yields their base address, what do you obtain on mentioning the structure name?

Ans:- The entire structure itself and not it’s base address.

1. What is main returning in the following program

struct transaction

{

int sno;

char desc[30];

char dc;

float amount;

}

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

{

struct transaction t;

scanf(“%d %s %c %f”, &t.sno, t.desc, &t.dc, &t.amount);

printf(“%d %s %c %f”, t.sno, t.desc, t.dc, t.amount);

}

Ans:- A missing semicolon at the end of the structure declaration is causing main to be declared as returning a structure.

1. O/p?

main()

{

struct a

{

category : 5;

scheme : 4;

};

printf(“size= %d”, sizeof(struct a));

}

Ans:- size=2

Since we have used bit fields in the structure and the total number of bits is turning out to be more than 8 the sizeof the structure is being reported as 2 bytes.

1. What is the difference between a structure and a union?

Ans:- A union is essentially a structure in which all of the fields overlay each other; you can use only one field at a time. You can also write to one field and read from another, to inspect a type’s bit patterns or interpret them differently.

1. Is it necessary that the size of all elements in a union should be same?

Ans:- No. Union elements can be of different sizes. If so, size of the union is size of the longest element in the union. As against this the size of the structure is the sun of the size of it’s members. In both cases, the size may be increased by padding.

1. Point out the error, if any

main()

{

union a

{

int i;

char ch[2];

};

union a z1={512};

union a z2= {0, 2};

}

Ans:- The ANSI C standard allows an initializer for the first member of a union. There is no standard way of initializing any other member, hence the error in initializing z2.

Many proposals have been advanced to allow more flexible union initialization, but none has been adopted yet. If you still want to initialize different members of the union then you can define several variant copies of a union, with the members in different orders, so that you can declare and initialize the one having the appropriate first member as shown bellow.

Union a

{  
 int i;

char ch[2];

};

union b

{

char ch[2];

int i;

};

main()

{

union a z1={512};

union b z2={0,2};

}

1. What is the difference between an enumeration and a set of preprocessor #defines?

Ans:- There is hardly any difference between the two, except that the #define has a global effect (throughout the file) whereas the enumeration can have an effect local to the block if desired. Some advantages of enumerations are that the numeric values are automatically assigned whereas in #define we have to explicitly define them. A disadvantage is that we have no control over the sizes of enumeration variables.

1. Since enumerations have integral type and enumeration constants are of type int can we freely intermix them with other integral types, without errors?

Ans:- Yes.

1. Is there an easy way to print enumeration values symbolically?

Ans:- No. You can write a small function, one pre enumeration, to map an enumeration constant to a string, either by using a switch statement or by searching an array.

1. What is the use of bit fields in structure declaration?

Ans:- Bit-fields are used to save space in structures having several binary flags or other small fields. Note that the colon notation for specifying the size of a field in bits is valid only in structures and in unions; you cannot use this mechanism to specify the size of arbitrary variables.

1. Can we have an array of bit fields?

Ans:- No.

CHAPTER 12

**INPUT/OUTPUT**

1. What would be the output of the following program?

main()

{

int a=250;

printf(“%1d”, a);

}

Ans- 250

1. O/p?

main()

{

float a=3.15529

printf(“\n %6.2f”, a);

printf(“\n %6.3f”, a);

printf(“\n %5.4f”, a);

printf(“\n %2.1f”, a);

printf(“\n %0.0f”, a);

}

Ans:-

3.16

3.155

3.1553

3.2

3

1. In the following code

#include<stdio.h>

main()

{

FILE \*fp;

Fp=fopen(“trial”,”r”);

}

fp points to

1. The first character in the file.
2. A structure which contains a char pointer which points to the first character in the file.
3. The name of the file.
4. None of above.

Ans:- B.

1. Point to the error in the following code if any

#include”stdio.h”

main()

{

unsigned char ch;

FILE \*fp;

fp=fopen(“trial”, ”r”);

while( (ch = getc(fp) ) !=EOF)

printf(“%c”,ch);

fclose(fp);

}

Ans:- EOF has been defined as #define EOF –1 in the file stdio.h and an unsigned character ranges from 0 to 255 hence when EOF is read from the file it can not be accommodated in ch. Solution is to declare ch as an integer.

1. Point out error if any in the following code.

#include”stdio.”

main()

{

unsigned char;

FILE \*fp;

fp= fopen(“trial”, “r”);

if(!fp)

{

printf(“\n Unable to open file”);

exit();

}

fclose(fp);

}

Ans:- No error.

1. If the file contains the line “I am a boy\r\n” then on reading this line into the array str using fgets( ) what would str contains?
   1. “I am a boy\r\n\0”
   2. “I am a boy\r\0”
   3. “I am a boy\n\0”
   4. “I am a boy”

Ans:- C.

1. Point out error if any in the following code

#include”stdio.h”

main()

{

FILE \*fp;

fp=fopen(“trial”, “r”);

fseek(fp, 20, SEEK\_SET);

fclose(fp);

}

Ans:- Instead of 20 use 20L since fseek( ) needs a long offset value.

1. To print out a and b given bellow, which printf statement would you use?

float a=3.14;

double b=3.14;

1. printf(“%f %F”, a, b);
2. printf(“%Lf %f”, a, b);
3. printf(“%Lf %Lf”, a, b);
4. printf(“%f %Lf”, a, b);

Ans:- A. It is possible to print a double using %f.

1. To scan a and b given bellow, which scanf( ) statement would you use?

float a;

double b;

1. scanf(“%f %f”, &a, &b);
2. scanf(“%Lf %Lf”, &a, &b);
3. scanf(“%f %Lf”, &a, &b);
4. scanf(“%f %lf”, &a, &b);

Ans:- D.

1. Point out error if any in the following code.

#include “stdio.h”

main( )

{

FILE \*fp;

char str[80];

fp= fopen(“trial’, “r”);

while(!eof(fp) )

{

fgets(str, 80, fp);

puts(str);

}

fclose(fp);

}

Ans:- The last line from the file trial should be read twice. To avoid this use

While(fgets(str, 80, fp)!= NULL)

Puts(str);

1. Point out error if any in the following code

#include “stdio.h”

main( )

{

char ch;

int i;

scanf(“%c”, &i);

scanf(“%d”, &ch);

printf(“%c %d”, ch, i);

}

Ans:- You would not get a chance to supply a character for the second scanf( ) statement. Solution is to precede the second scanf( ) statement with the following statement

fflush(stdin);

This would flush out the enter hit for the previous scanf( ) to be flushed out from the input stream, i.e. keyboard.

1. O/?

main()

{

printf(“\n%%%%”);

}

Ans:- %%

1. Point out error if any in the following code

#include”stdio.h”

main()

{

FILE \*fp;

fp=fopen(“c:\tc\trial”, “w”);

if(!fp)

exit( );

fclose( );

}

Ans:- The path of the file should have been written a “c:\\tc\\trial”.

1. Would the following code works? If yes what would be the output?

main()

{

int n=5;

printf(“\n n=%\*d”, n, n);

}

Ans:- Yes

N= 5

15 What is the \* in a printf( ) of Q14 above mean?

Ans:- It indicates that an integer value from the argument list will be used for field width. In the argument list the width preceeds the value to be printed. In this case the format specifier becomes %5d.

16.Can we specify variable field width in a scanf( ) format string?

Ans:- No. A \* in scanf( ) format string after a %sign is used for assignment suppression. That is, the current input field is scaned but not stored.

17. To tackle a double in printf( ) we can use %f, whereas in scanf( ) we should use %lf.

Ans:- True.

18. Out of fgets( ) and gets( ) which function is safe to use?

Ans:- fgets( ), because unlike fgets( ), gets( ) cannot be told the size of the buffer into which the string supplied would be stored. As a result there is always a possibility of overflow of buffer.

19. A file written in text mode can be read back in binary mode.

Ans:- False.

20. We should not read after a write to a file without an intervening call to fflush( ), fseek( ) or rewind( ).

Ans:- True.

CHAPTER 13

**COMMAND LINE ARGUMENTS**

1. what do the c and v in argc and argv stand for?

Ans:- Count of arguments and vector (array) of arguments.

1. According to ANSI specifications which is correct way of declaring main( ) when it receives command line arguments?
   1. main(int argc, char \*argv[])
   2. main(argc, argv)

int argc; char \*argv[];

* 1. main( )

{

int argc; char \*argv[];

}

* 1. None of above.

Ans:- A

3.O/p?

/\*sample.\*/

main(int argc, char \*\*argv)

{

argc=argc-(argc-1);

printf(“%s”, argv[argc-1]);

}

Ans:- C:\SAMPLE.EXE

1. If different command arguments are supplied at different times would the output of the following program change?

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

{

printf(“%d”,argv[argc]);

}

Ans:- No.

1. If the following program myprog.c is run from the command line as

Myprog 1 2 3

What would be the output?

Main(int argc, char \*argv[])

{

int x;

for(x=0; x<argc; x++)

printf(“%d”, argv[x]);

}

Ans:- C:\MYPROG.EXE 1 2 3

1. If the following program muprog.c is run from the command line as

myprog 1 2 3

What would be the o/p?

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

{

int x;

x=argv[1]+argv[2]+argv[3];

printf(“%d”, x);

}

a.123

b.6

c.Error

d.”123”

Ans:- C

1. If the following program myprog.c is run from the command line as

myprog 1 2 3

What would be the o/p?

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

{

int x, j=0;

for(x=0; x<argc; x++)

j=j+atoi(argv[x]);

printf(“5d”, j);

}

1. 123
2. 6
3. Error
4. “123”

Ans:- B. When atoi( ) tries to convert arv[0] to a number it can not do so, argv[0] being the file name and hence returns a zero

1. If the following program myprog.c is run from the command line as

Myprog one two three

What would be the o/p?

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

{

printf(“%s”, \*++argv);

}

Ans:- one

1. If the following program myprog.c is run from the command line as

myprog one two three

What would be the o/p?

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

{

printf(“%s”, ++\*\*++argv);

}

Ans:- p

1. The variables argc and argv are always local to main?

Ans:- True.

1. The maximum combined length of the command line arguments including spaces between adjacent arguments is

(a:- 128 chars-----b:- 256 chars-----c:- 67 chars------d:- may vary from o.s. to o.s.)

Ans:- D

1. O/p?

main(int argc, char \*argv[], char –env[])

{

int x;

for(x=0; x<arc; x++)

printf(“%s”, env[x]);

}

1. List of all environment variables.
2. List of all command line arguments.
3. Error
4. NULL.

Ans:- B.