

C - Programming

Scope of a variable :-

1) Static Scoping (default)

2) Dynamic

Ex: int a=55;
main()

mem. at compile time

{ int a=5;
B();
}

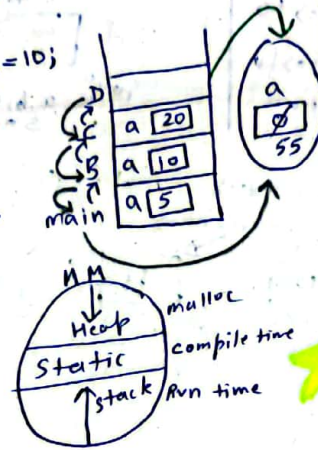
{ int a=10;
C();
}

C()
{ int a=20;
D();
}

D()
{ int a=20;
Pf(a);
}

Dynamic scoping : o/p = 20

Static scoping : o/p = 55



Static variable :

Ex: main()

```

{
    static int var = 5;
    if (var) { pf(var--);
              main();
            }
}
    
```



main()

1. x

2. if(5)

3. ✓

4. main()

1. x

2. if(4)

3. ✓

4. main()

1. x

2. if(3)

3. ✓

4. main()

1. x

2. if(2)

3. ✓

4. main()

1. x

2. if(1)

3. ✓

4. main()

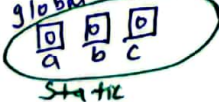
1. x

2. if(0)

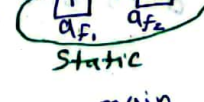
3. ✓

4. main()

Notes) Also possible



Static Var.

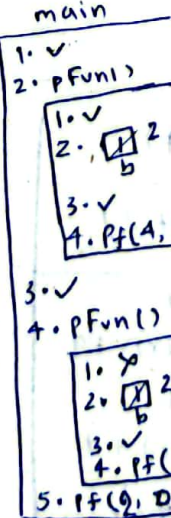


Ex: int a, b, c = 0;

```

main()
{
    static int a = 1;
    pf(a);
    a++;
    pf(a);
    pf("in %d %d", a, b);
}
    
```

o/p: 4, 2, 6, 2, 2, 0

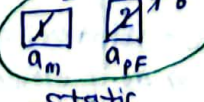


global var



Static

Static Var.



Static

Note: auto int a=1; register int a=2; => inside stack
i.e. same as int a=2;

#) Extern Variable: variable sharing memory with global variable

Ex: int a=5;
main() { extern int a;
a=20;
pf(a);
}



Note: Global extern variable will allocate memory if it is initialized, but not local extern variable.

Note: Extern keyword power is that u can create 1 lakh local variables with same name.

Associativity & Precedence :

() [] -> .	pointer	L to R
! ~ ++ -- + - * (type) sizeof		R to L
* / %		L to R
+ -		L to R
<< >>		L to R
< <= > >=		L to R
== !=		L to R
& Bitwise AND		L to R
^ Bitwise XOR		L to R
Bitwise OR		L to R
&&		L to R
		L to R
? :		R to L
= += -= *= /= %= &= ^= = <<= >>=		R to L

#) a[3][5] = {10, 20, 30, ..., 150}

0	1	2	3	4
1000	1002	1004	1006	1008
1010	1012	1014	1016	1018
1020	1022	1024	1026	1028
1030	1032	1034	1036	1038
1040	1042	1044	1046	1048
1050	1052	1054	1056	1058
1060	1062	1064	1066	1068
1070	1072	1074	1076	1078
1080	1082	1084	1086	1088
1090	1092	1094	1096	1098
1100	1102	1104	1106	1108
1110	1112	1114	1116	1118
1120	1122	1124	1126	1128
1130	1132	1134	1136	1138
1140	1142	1144	1146	1148
1150	1152	1154	1156	1158

a => 1000
a+1 => 1002 (1 row skip)
a+2 => 1004 (2 row skip)
a+3 => 1006 (3 row skip)
a+4 => 1008 (4 row skip)
a+5 => 1010 (5 row skip)
a+6 => 1012 (6 row skip)
a+7 => 1014 (7 row skip)
a+8 => 1016 (8 row skip)
a+9 => 1018 (9 row skip)
a+10 => 1020 (10 row skip)
a+11 => 1022 (11 row skip)
a+12 => 1024 (12 row skip)
a+13 => 1026 (13 row skip)
a+14 => 1028 (14 row skip)
a+15 => 1030 (15 row skip)
a+16 => 1032 (16 row skip)
a+17 => 1034 (17 row skip)
a+18 => 1036 (18 row skip)
a+19 => 1038 (19 row skip)
a+20 => 1040 (20 row skip)
a+21 => 1042 (21 row skip)
a+22 => 1044 (22 row skip)
a+23 => 1046 (23 row skip)
a+24 => 1048 (24 row skip)
a+25 => 1050 (25 row skip)
a+26 => 1052 (26 row skip)
a+27 => 1054 (27 row skip)
a+28 => 1056 (28 row skip)
a+29 => 1058 (29 row skip)
a+30 => 1060 (30 row skip)
a+31 => 1062 (31 row skip)
a+32 => 1064 (32 row skip)
a+33 => 1066 (33 row skip)
a+34 => 1068 (34 row skip)
a+35 => 1070 (35 row skip)
a+36 => 1072 (36 row skip)
a+37 => 1074 (37 row skip)
a+38 => 1076 (38 row skip)
a+39 => 1078 (39 row skip)
a+40 => 1080 (40 row skip)
a+41 => 1082 (41 row skip)
a+42 => 1084 (42 row skip)
a+43 => 1086 (43 row skip)
a+44 => 1088 (44 row skip)
a+45 => 1090 (45 row skip)
a+46 => 1092 (46 row skip)
a+47 => 1094 (47 row skip)
a+48 => 1096 (48 row skip)
a+49 => 1098 (49 row skip)
a+50 => 1100 (50 row skip)
a+51 => 1102 (51 row skip)
a+52 => 1104 (52 row skip)
a+53 => 1106 (53 row skip)
a+54 => 1108 (54 row skip)
a+55 => 1110 (55 row skip)
a+56 => 1112 (56 row skip)
a+57 => 1114 (57 row skip)
a+58 => 1116 (58 row skip)
a+59 => 1118 (59 row skip)
a+60 => 1120 (60 row skip)
a+61 => 1122 (61 row skip)
a+62 => 1124 (62 row skip)
a+63 => 1126 (63 row skip)
a+64 => 1128 (64 row skip)
a+65 => 1130 (65 row skip)
a+66 => 1132 (66 row skip)
a+67 => 1134 (67 row skip)
a+68 => 1136 (68 row skip)
a+69 => 1138 (69 row skip)
a+70 => 1140 (70 row skip)
a+71 => 1142 (71 row skip)
a+72 => 1144 (72 row skip)
a+73 => 1146 (73 row skip)
a+74 => 1148 (74 row skip)
a+75 => 1150 (75 row skip)
a+76 => 1152 (76 row skip)
a+77 => 1154 (77 row skip)
a+78 => 1156 (78 row skip)
a+79 => 1158 (79 row skip)
a+80 => 1160 (80 row skip)
a+81 => 1162 (81 row skip)
a+82 => 1164 (82 row skip)
a+83 => 1166 (83 row skip)
a+84 => 1168 (84 row skip)
a+85 => 1170 (85 row skip)
a+86 => 1172 (86 row skip)
a+87 => 1174 (87 row skip)
a+88 => 1176 (88 row skip)
a+89 => 1178 (89 row skip)
a+90 => 1180 (90 row skip)
a+91 => 1182 (91 row skip)
a+92 => 1184 (92 row skip)
a+93 => 1186 (93 row skip)
a+94 => 1188 (94 row skip)
a+95 => 1190 (95 row skip)
a+96 => 1192 (96 row skip)
a+97 => 1194 (97 row skip)
a+98 => 1196 (98 row skip)
a+99 => 1198 (99 row skip)
a+100 => 1200 (100 row skip)
a+101 => 1202 (101 row skip)
a+102 => 1204 (102 row skip)
a+103 => 1206 (103 row skip)
a+104 => 1208 (104 row skip)
a+105 => 1210 (105 row skip)
a+106 => 1212 (106 row skip)
a+107 => 1214 (107 row skip)
a+108 => 1216 (108 row skip)
a+109 => 1218 (109 row skip)
a+110 => 1220 (110 row skip)
a+111 => 1222 (111 row skip)
a+112 => 1224 (112 row skip)
a+113 => 1226 (113 row skip)
a+114 => 1228 (114 row skip)
a+115 => 1230 (115 row skip)
a+116 => 1232 (116 row skip)
a+117 => 1234 (117 row skip)
a+118 => 1236 (118 row skip)
a+119 => 1238 (119 row skip)
a+120 => 1240 (120 row skip)
a+121 => 1242 (121 row skip)
a+122 => 1244 (122 row skip)
a+123 => 1246 (123 row skip)
a+124 => 1248 (124 row skip)
a+125 => 1250 (125 row skip)
a+126 => 1252 (126 row skip)
a+127 => 1254 (127 row skip)
a+128 => 1256 (128 row skip)
a+129 => 1258 (129 row skip)
a+130 => 1260 (130 row skip)
a+131 => 1262 (131 row skip)
a+132 => 1264 (132 row skip)
a+133 => 1266 (133 row skip)
a+134 => 1268 (134 row skip)
a+135 => 1270 (135 row skip)
a+136 => 1272 (136 row skip)
a+137 => 1274 (137 row skip)
a+138 => 1276 (138 row skip)
a+139 => 1278 (139 row skip)
a+140 => 1280 (140 row skip)
a+141 => 1282 (141 row skip)
a+142 => 1284 (142 row skip)
a+143 => 1286 (143 row skip)
a+144 => 1288 (144 row skip)
a+145 => 1290 (145 row skip)
a+146 => 1292 (146 row skip)
a+147 => 1294 (147 row skip)
a+148 => 1296 (148 row skip)
a+149 => 1298 (149 row skip)
a+150 => 1300 (150 row skip)

a[15] = {10, 20, ..., 150}

0	10	1000
1	20	1002
2	30	1004
3	40	1006
4	50	1008
5	60	1010
6	70	1012
7	80	1014
8	90	1016
9	100	1018
10	110	1020
11	120	1022
12	130	1024
13	140	1026
14	150	1028

Note: Static Scoping = early binding

Dynamic = late binding

Note: Wrong Practice in pointers:-
int a = address x
int xa = data x

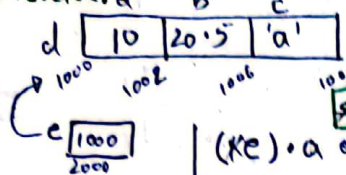
a => 1000
a+1 => 1002
a+3 => 1006

a => 1000
a+1 => 1002
a+3 => 1006

Note: In ASCII
A = 65 a = 97

#> Structure & Union:

Structure: a b c

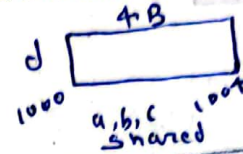


```
struct node {
    int a;
    float b;
    char c;
};
d.a = 10;
d.b = 20.5;
d.c = 'a';
struct node *e = d;
```

($*e$).a or $e \rightarrow a \Rightarrow 10$
 $*e \Rightarrow \text{error}$
 $e+1 \Rightarrow 1007$

$d \Rightarrow 1000$
 $\text{pf}(\%d, d) \Rightarrow 10$
 $\text{pf}(\%d, *e) \Rightarrow 10$
 $\text{pf}(e) \Rightarrow 1000$

Union: Stores only one variable at a time, when you store another variable, the previous one gets deleted



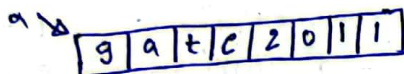
2

#> Arrays:

> char a[] = "gate2011";

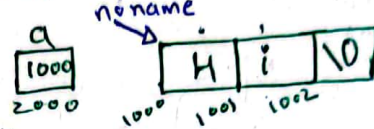


char a[] = {'g', 'a', 't', 'e', '2', '0', '1', '1'};



> $*a+2 = 'g' \Rightarrow \text{possible}$
 $a = a+2 \Rightarrow \text{Not-possible}$
 <Address Constant>

char *a = "Hi";



$a = a+2 \Rightarrow \text{possible}$
 $*a+2 = 'g' \Rightarrow \text{May not possible}$
 <String constant>

> $\text{strlen}(a) = 8$
 size of (a) = 9

> $a[2]$ is $*a+2$ > $a[i]$ is $*a+i$ is $*(a+i)$ is $i[a]$

> $a[i][j]$ is $*(a+i+j)$ > $a[i][j][k]$ is $*(a+i+j+k)$

> $if(0)$
 $if(0.0)$
 $if('0')$
 $if(NULL)$ } cond.ⁿ fail

> char a = 'e';
 $a[0] \Rightarrow [101] \Rightarrow \text{ASCII binary}$

> float *b; b[2000] 3000
 $b++ \Rightarrow b[2001]$
 $b+2 \Rightarrow b[2002]$

> In 2d Array $a[1]$ is base address of 1st row, a is base address of whole 2d array.

> Ex: int a[5][3][7]; BA = 1000



$a \Rightarrow 1000$ (i.e. 000)
 $*a \Rightarrow 1000$ (i.e. 0th 2d selected)
 $**a \Rightarrow 1000$ (0th 2d, 0th row selected)
 $***a \Rightarrow \text{value}(a[0][0][0])$

$a \Rightarrow 1000$
 $*a \Rightarrow 1000$ (0th 2d selected)
 $*a+5 \Rightarrow 1070$
 rows

$**a \Rightarrow 1000$ (0th 2d, 0th row selected)
 $**a+5 \Rightarrow 1010$
 elements

$a \Rightarrow 1000$
 $a+1 \Rightarrow 1042$ (1st 2d's skip)
 $a+3 \Rightarrow 1126$ (3 2d's skip)
 $*a+3 \Rightarrow 1126$ (3rd 2d selected)

$a[2]+9 \Rightarrow *(a+2)+9$
 $= *(1000+2 \times 21 \times 2)+9$
 $= *(1084)+9$
 $= 2^{\text{nd}} \text{ 2d selected } + 9$
 $= 1084 + 9 \times 7 \times 2$
 $= 1210$

$a \Rightarrow 1000$
 $*a \Rightarrow 1000$
 $*a+1 \Rightarrow 1000 + 5 \times 3 \times 7 \times 2$
 $= 1210$
 1 3d's skip

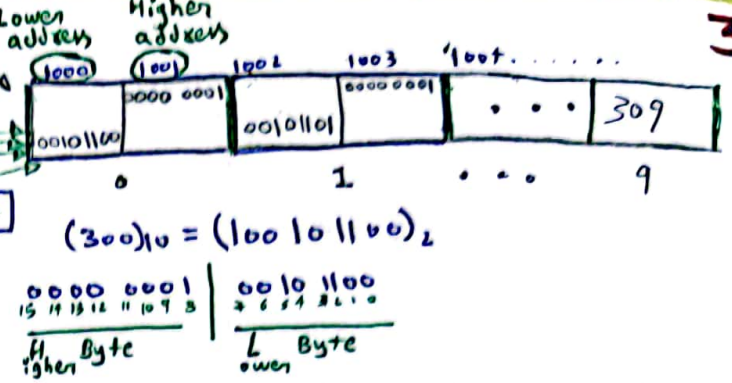
3. $\text{Pointer}_1 + n$ or $*\text{Pointer}_2 \Rightarrow \text{Error}$

Note: 1) $\text{Pointer Var} \pm \text{Integer constant} \Rightarrow \checkmark \Rightarrow \text{Skipping elements}$ 2) $\text{Pointer}_1 - \text{Pointer}_2 \Rightarrow \checkmark \Rightarrow \text{No. of elements b/w the pointers}$
 $P_2 - P_1 \Rightarrow \text{No. of elements from } P_1 \text{ before } P_2$
 (provided: 1) $P_2 \geq P_1$
 2) P_1, P_2 points same array)

int *p;
 $P+5 \Rightarrow \text{Skip 5 elements in forward}$
 $P-5 \Rightarrow \text{Skip 5 elements in backward}$

#>Typecasting in pointers:

- ① `int a[10] = { 300, 301, 302, ... 309 };`
- ② `int *b = a;`
- ③ `char *c = (char *)a;`
- ④ `float *d = (float *)a;`
- ⑤ `void *e = a;`
- ⑥ `int *f = (int *)e;`
- ⑦ `b = d;` // Error
- ⑧ `b = c;` // No Error



Little Endian \Rightarrow Lower addresses contain Lower Byte

$(300)_{10} = (00101100)_2$

Higher Byte: 0000 0001
Lower Byte: 0010 1100

$b \Rightarrow 1000$
 $*b \Rightarrow 300$
 $b+1 \Rightarrow 1002$

$c \Rightarrow 1000$
 $*c \Rightarrow 44$
 $c+1 \Rightarrow 1001$

$d \Rightarrow 1000$
 $*d \Rightarrow 48$ data from 1000 onwards in IEEE 754
 $d+1 \Rightarrow 1001$

$e \Rightarrow 1000$
 $*e \Rightarrow \text{Error}$
 $e+1 \Rightarrow \text{Error}$

#>Parameter passing techniques:-

1.) Call by value:- (call by copy)

main() `a` `b` `swap(int c, int d)`

`int a=10, b=20;`
`Pf(a,b);`
`Swap(a,b);`
`Pf(a,b);`

`int t;`
`t=c;`
`c=d;`
`d=t;`

o/p: 10,20
10,20

other names:-
call by name
call by need

2.) Call by reference:-

main() `a` `b`

`int a=10, b=20;`
`Pf(a,b);`
`Swap(a,b);`
`Pf(a,b);`

`Swap(int c, int d)`

`int t;`
`t=*c;`
`*c=*d;`
`*d=t;`

o/p: 10,20
20,10

used when we want to pass very less no. of parameters.

used when we want to pass very large no. of parameters.

Note: C lang. is C & C++ bec.

1. Variable declared before use
2. Matching formal & actual parameters of fcn

PDA can't handle

Here, How a compiler will behave, which supports call by reference:-
That lang. will himself keep '&' in `swap(a,b)` & '*' in `swap(c,d)`.

Note:)

`int a=3, b=5;`
`unsigned int c=0;`
`x = (a-b) > c ? 1 : 0;`

Here, `(3-5) > 0`
`2 > 0` [unsigned int]

so, `x = 1`

Note:)

`*p1` `*p2`

`Pf(p2-p1);`

$\Rightarrow \frac{2008-2000}{8} = 1 = \text{o/p}$

Note:)

define `F(a,b) (a*b)`

`a, b` are formal parameters

Note:)

`x` `11` `-11`

`~x` `-12` `10`

बड़ा हो & sign change
छोटा हो & sign change

\sim is Bitwise complement (1's)

bec. `11 = 0000 1011`
`-12 = 1111 0100` is