CHAPTER 8

MORE ABOUT POINTERS

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

Ans: NO.

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

Ans: In files stdio.h & stddef.h

3. 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.

4. 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 .

5. 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

7. 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.

8. 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