PASCAL - MEMORY MANAGEMENT

This chapter explains dynamic memory management in Pascal. Pascal programming language provides several functions for memory allocation and management.

Allocating Memory Dynamically

While doing programming, if you are aware about the size of an array, then it is easy and you can define it as an array. For example, to store a name of any person, it can go max 100 characters so you can define something as follows −

```
var
name: array[1..100] of char;
```

But now, let us consider a situation, where you have no idea about the length of the text you need to store, for example, you want to store a detailed description about a topic. Here, we need to define a pointer to string without defining how much memory is required.

Pascal provides a procedure **new**to create pointer variables.

```
program exMemory;
var
name: array[1..100] of char;
description: ^string;

begin
   name:= 'Zara Ali';

   new(description);
      if not assigned(description) then
         writeln(' Error - unable to allocate required memory')
      else
         description^ := 'Zara ali a DPS student in class 10th';
   writeln('Name = ', name );
   writeln('Description: ', description^ );
end.
```

When the above code is compiled and executed, it produces the following result −

```
Name = Zara Ali
Description: Zara ali a DPS student in class 10th
```

Now, if you need to define a pointer with specific number of bytes to be referred by it later, you should use the **getmem** function or the **getmem** procedure, which has the following syntax −

```
procedure Getmem(
   out p: pointer;
   Size: PtrUInt
);

function GetMem(
   size: PtrUInt
):pointer;
```

In the previous example, we declared a pointer to a string. A string has a maximum value of 255 bytes. If you really don't need that much space, or a larger space, in terms of bytes, *getmem* subprogram allows specifying that. Let us rewrite the previous example, using *getmem* −

```
program exMemory;
var
name: array[1..100] of char;
```

```
description: ^string;

begin
   name:= 'Zara Ali';

   description := getmem(200);
      if not assigned(description) then
         writeln(' Error - unable to allocate required memory')
      else
         description^ := 'Zara ali a DPS student in class 10th';
   writeln('Name = ', name );
   writeln('Description: ', description^ );

   freemem(description);
end.
```

When the above code is compiled and executed, it produces the following result −

```
Name = Zara Ali
Description: Zara ali a DPS student in class 10th
```

So, you have complete control and you can pass any size value while allocating memory unlike arrays, where once you defined the size cannot be changed.

Resizing and Releasing Memory

When your program comes out, operating system automatically releases all the memory allocated by your program, but as a good practice when you are not in need of memory anymore, then you should release that memory.

Pascal provides the procedure **dispose** to free a dynamically created variable using the procedure **new.** If you have allocated memory using the **getmem** subprogram, then you need to use the subprogram **freemem** to free this memory. The *freemem* subprograms have the following syntax −

```
procedure Freemem(
   p: pointer;
  Size: PtrUInt
);

function Freemem(
   p: pointer
):PtrUInt;
```

Alternatively, you can increase or decrease the size of an allocated memory block by calling the function *ReAllocMem*. Let us check the above program once again and make use of *ReAllocMem* and *freemem* subprograms. Following is the syntax for *ReAllocMem* −

```
function ReAllocMem(
   var p: pointer;
   Size: PtrUInt
):pointer;
```

Following is an example which makes use of *ReAllocMem* and *freemem* subprograms −

```
program exMemory;
var
name: array[1..100] of char;
description: ^string;
desp: string;

begin
   name:= 'Zara Ali';
   desp := 'Zara ali a DPS student.';

   description := getmem(30);
```

```
      if not assigned(description) then
         writeln('Error - unable to allocate required memory')
      else
         description^ := desp;

   (* Suppose you want to store bigger description *)
   description := reallocmem(description, 100);
   desp := desp + ' She is in class 10th.';
   description^:= desp;

   writeln('Name = ', name );
   writeln('Description: ', description^ );

   freemem(description);
end.
```

When the above code is compiled and executed, it produces the following result −

```
Name = Zara Ali
Description: Zara ali a DPS student. She is in class 10th
```

Memory Management Functions

Pascal provides a hoard of memory management functions that is used in implementing various data structures and implementing low-level programming in Pascal. Many of these functions are implementation dependent. Free Pascal provides the following functions and procedures for memory management −

S.N	Function Name & Description
1	**function Addr*X*: *TAnytype*:Pointer;** Returns address of variable
2	**function Assigned*P*: *Pointer*:Boolean;** Checks if a pointer is valid
3	**function CompareByte*constbuf*1; *constbuf*2; *len*: *SizeInt*:SizeInt;** Compares 2 memory buffers byte per byte
4	**function CompareChar*constbuf*1; *constbuf*2; *len*: *SizeInt*:SizeInt;** Compares 2 memory buffers byte per byte
5	**function CompareDWord*constbuf*1; *constbuf*2; *len*: *SizeInt*:SizeInt;** Compares 2 memory buffers byte per byte
6	**function CompareWord*constbuf*1; *constbuf*2; *len*: *SizeInt*:SizeInt;** Compares 2 memory buffers byte per byte
7	

function Cseg: Word;

Returns code segment

8

procedure Dispose*P*: *Pointer*;

Frees dynamically allocated memory

9

procedure Dispose*P*: *TypedPointer*; *Des*: *TProcedure*;

Frees dynamically allocated memory

10

function Dseg: Word;

Returns data segment

11

procedure FillByte*varx*; *count*: *SizeInt*; *value*: *Byte*;

Fills memory region with 8-bit pattern

12

procedure FillChar*varx*; *count*: *SizeInt*; *Value*: *Byte*|*Boolean*|*Char*;

Fills memory region with certain character

13

procedure FillDWord*varx*; *count*: *SizeInt*; *value*: *DWord*;

Fills memory region with 32-bit pattern

14

procedure FillQWord*varx*; *count*: *SizeInt*; *value*: *QWord*;

Fills memory region with 64-bit pattern

15 **procedure FillWord***varx*; *count*: *SizeInt*; *Value*: *Word*;

Fills memory region with 16-bit pattern

16

procedure Freemem*p*: *pointer*; *Size*: *PtrUInt*;

Releases allocated memory

17

procedure Freemem*p*: *pointer*;

Releases allocated memory

18

procedure Getmem*outp*: *pointer*; *Size*: *PtrUInt*;

Allocates new memory

19

procedure Getmem*outp*: *pointer*;

Allocates new memory

20

procedure GetMemoryManager*varMemMgr*: *TMemoryManager*;

Returns current memory manager

21

function High*Arg*: *TypeOrVariable*:**TOrdinal;**

Returns highest index of open array or enumerated

22

function IndexByte*constbuf*; *len*: *SizeInt*; *b*: *Byte*:**SizeInt;**

Finds byte-sized value in a memory range

23

function IndexChar*constbuf*; *len*: *SizeInt*; *b*: *Char*:**SizeInt;**

Finds char-sized value in a memory range

24

function IndexDWord*constbuf*; *len*: *SizeInt*; *b*: *DWord*:**SizeInt;**

Finds DWord-sized $32-bit$ value in a memory range

25

function IndexQWord*constbuf*; *len*: *SizeInt*; *b*: *QWord*:**SizeInt;**

Finds QWord-sized value in a memory range

26

function Indexword*constbuf*; *len*: *SizeInt*; *b*: *Word*:**SizeInt;**

Finds word-sized value in a memory range

27

function IsMemoryManagerSet: Boolean;

Is the memory manager set

28

function Low*Arg*: *TypeOrVariable*:**TOrdinal;**

Returns lowest index of open array or enumerated

29

procedure Move*constsource*; *vardest*; *count*: *SizeInt*;

Moves data from one location in memory to another

30

procedure MoveChar0*constbuf*1; *varbuf*2; *len*: *SizeInt*;

Moves data till first zero character

31

procedure New*varP*: *Pointer*;

Dynamically allocate memory for variable

32

procedure New*varP*: *Pointer*; *Cons*: *TProcedure*;

Dynamically allocates memory for variable

33

function Ofs*varX*:**LongInt;**

Returns offset of variable

34

function ptr*sel*: *LongInt*; *off*: *LongInt*:**farpointer;**

Combines segment and offset to pointer

35

function ReAllocMem*varp*: *pointer*; *Size*: *PtrUInt*:**pointer;**

Resizes a memory block on the heap

36

function Seg*varX*:**LongInt;**

Returns segment

37

procedure SetMemoryManager*constMemMgr*: *TMemoryManager*;

Sets a memory manager

38

function Sptr: Pointer;

Returns current stack pointer

39

function Sseg: Word;

Returns stack segment register value