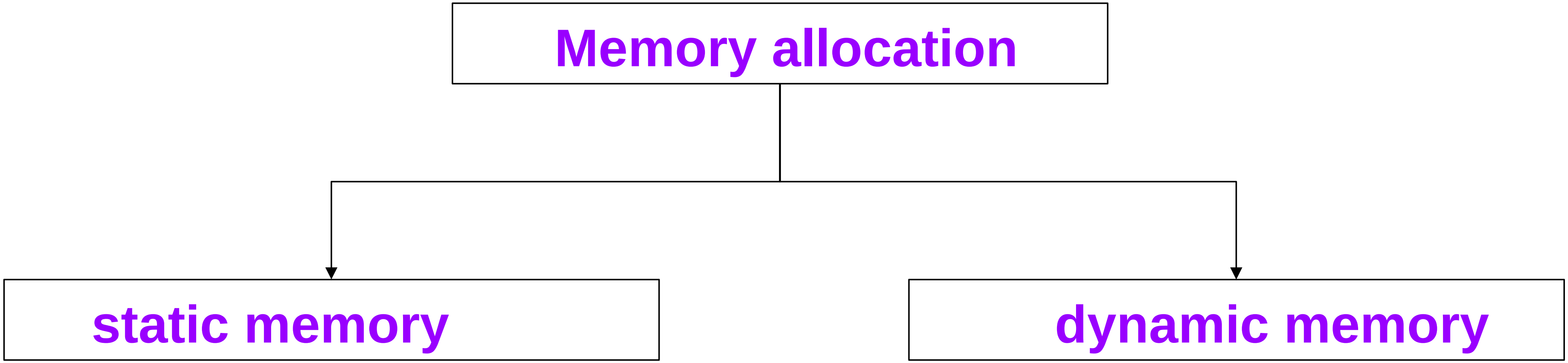
Dynamic Memory Allocation



Memory allocation

static memory

dynamic memory

1. it allocates before prog execution.
   (loadtime memory)

2. it is fixed memory.(it is not possible
   to increase or decrease the memory
   during runtime).

3. it is not possible to free the memory
   during runtime.

4. Faster in execution.

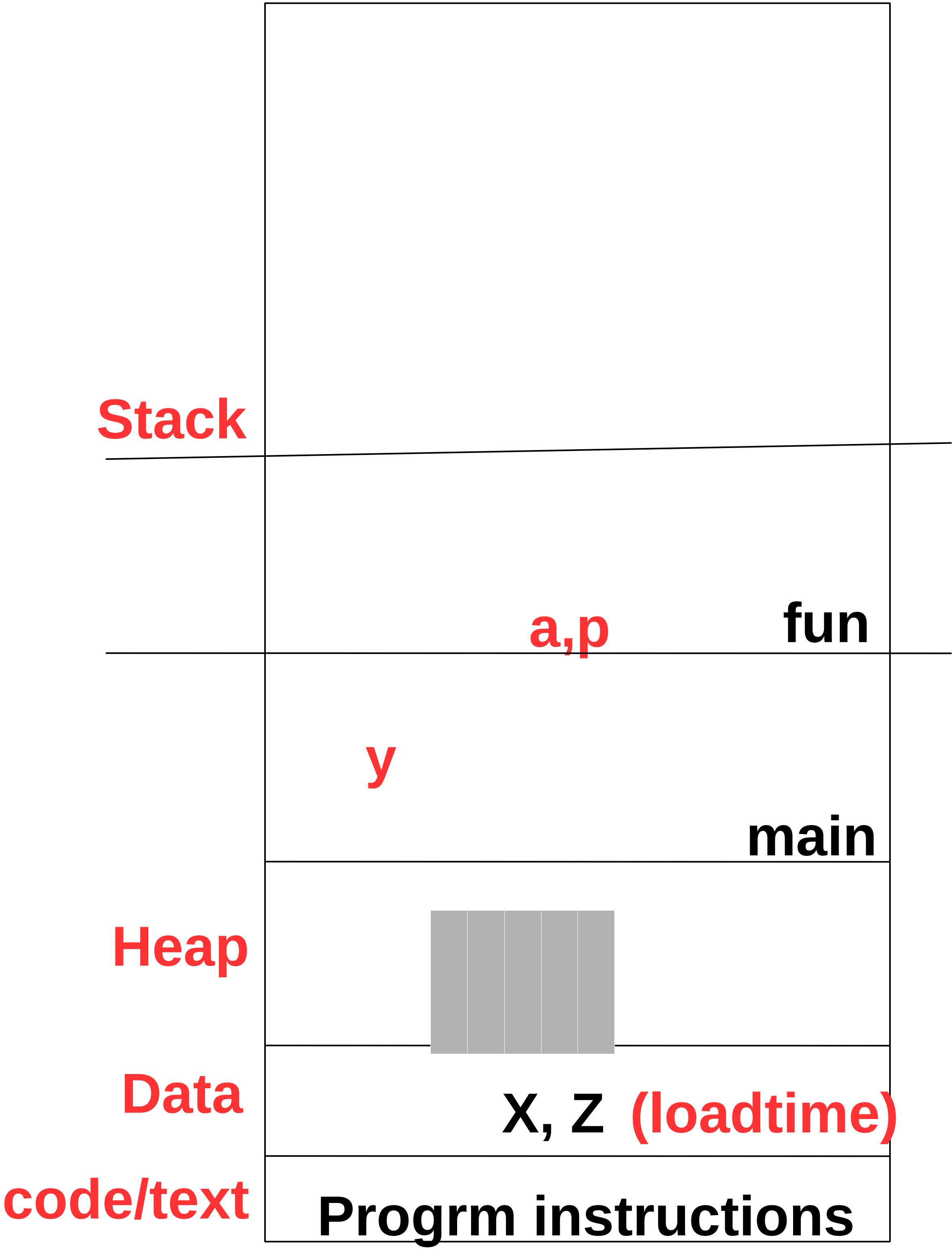
1. it allocates after prog execution.
   (runtime memory)

2. it is flexible memory.
   (it is possible to increase or
   decrease the memory).

3. it is possible to free the memory
   During runtime.

4. Slower in execution.

```
#include<stdio.h>
int x = 10;
main()
{
    int y = 20;
    fun();
}
void fun()
{
    static int z = 15;
    int a = 20;
    int *p = malloc(20);
}
```

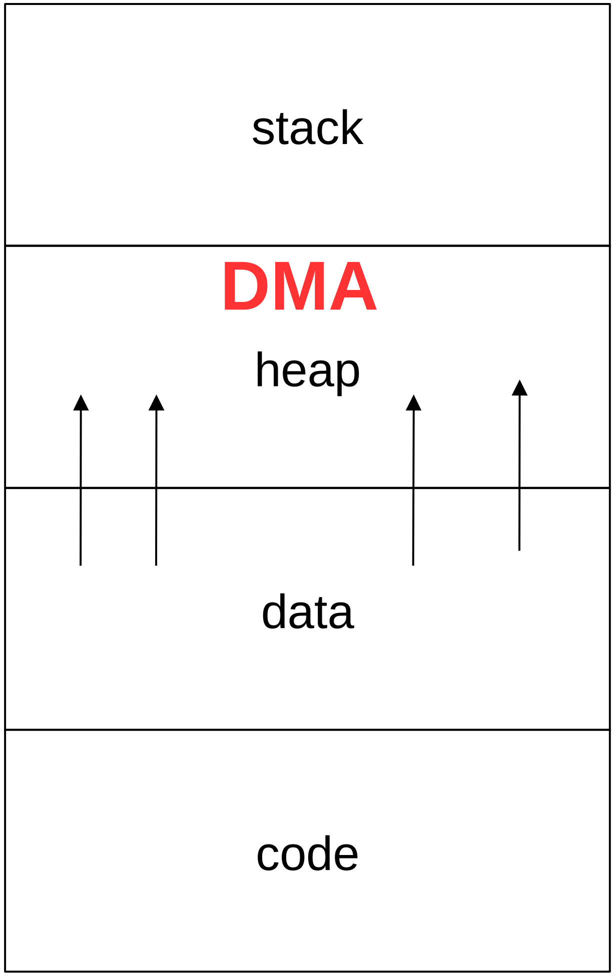


Heap : **It is an extension of data section.**

If we wants to allocate DMA , then there are 3 library functions.

1) malloc()
2) calloc()
3) realloc()

To de-allocate the memory
4) free()



SYNOPSIS

```
#include <stdlib.h>

void *malloc(size_t size);
```

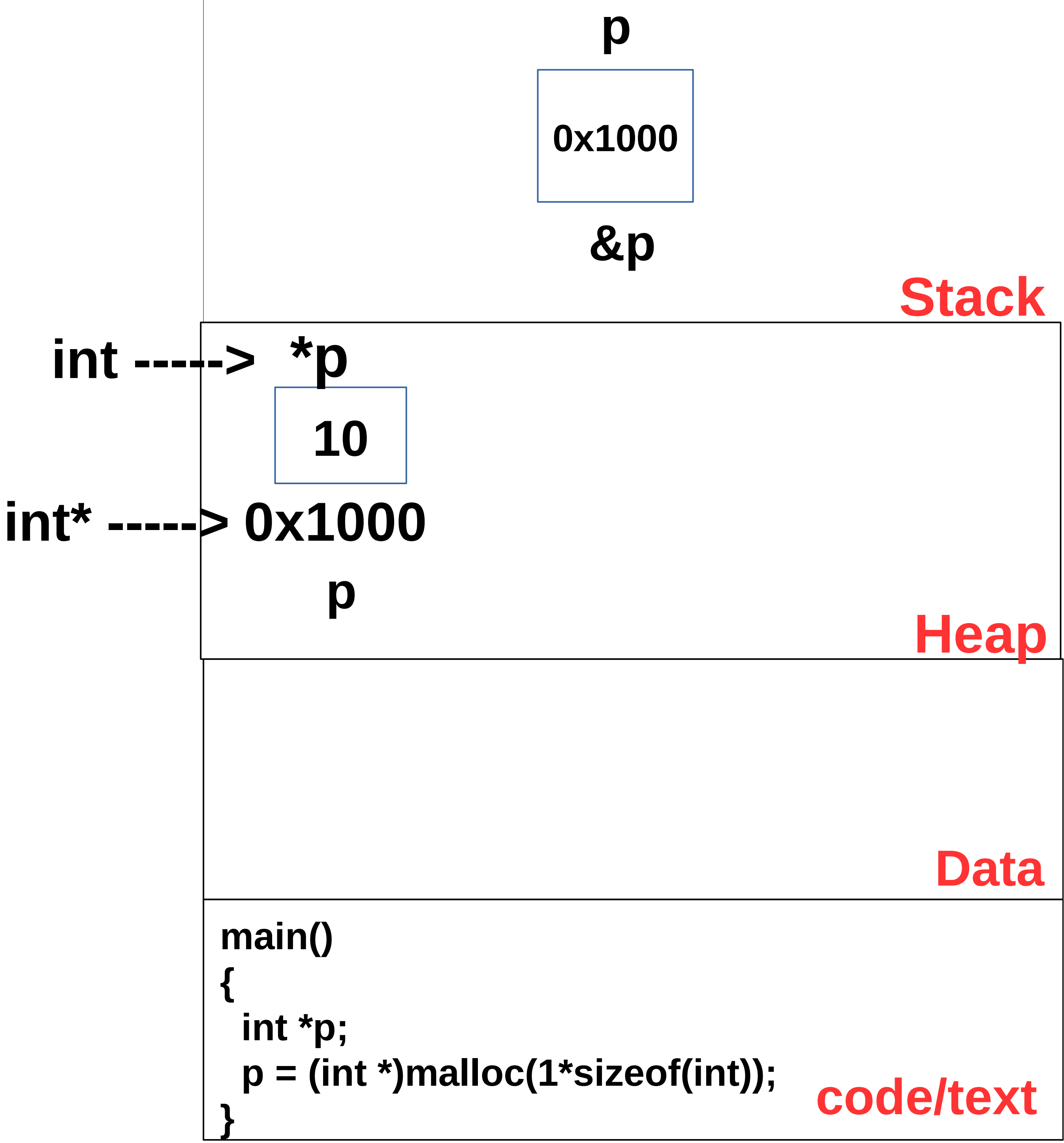
DESCRIPTION

The malloc() function allocates size bytes and returns a pointer to the allocated memory. The memory is not initialized. If size is 0, then malloc() returns either NULL, or a unique pointer value that can later be successfully passed to Free().

RETURN VALUES

Success : returns allocated memory base address.
Failure : returns NULL address.

```
//write a program to allocate memory dynamically for 1 integer.

#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *p = (int *)malloc(1*sizeof(int));

        if(p == NULL) {
        printf("failed to allocate DMA\n");
        return 0;
        }

        printf("Enter the value\n");
        scanf("%d",p);

        printf("*p = %d\n",*p);
}
```



p

0x1000

&p

Stack

int -----> *p

10

int* -----> 0x1000

p

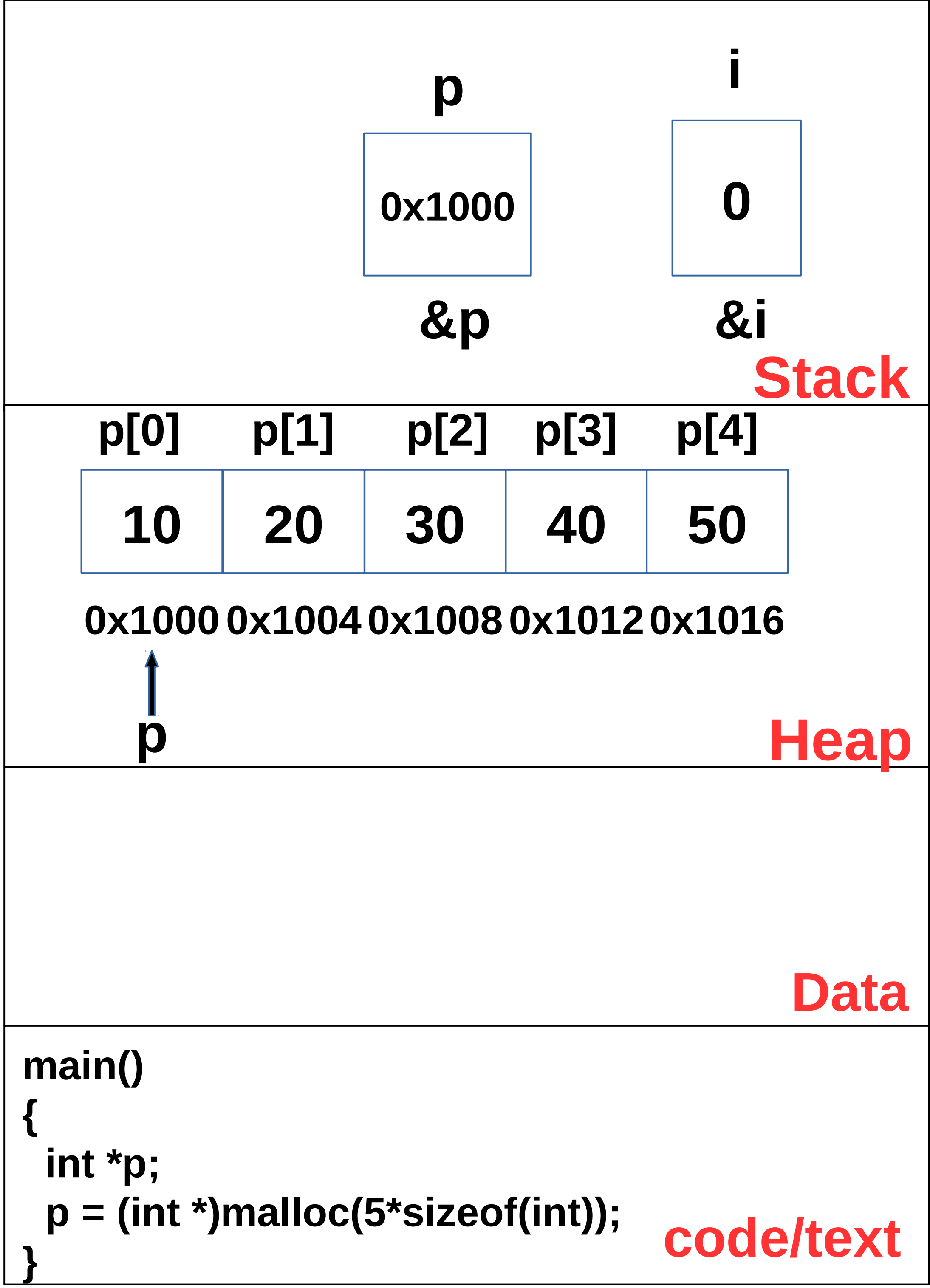
Heap

Data

```
main()
{
  int *p;
  p = (int *)malloc(1*sizeof(int));
}
```

code/text

```
#include<stdio.h>
main()
{
  int *p,i;
  p = (int *)malloc(5*sizeof(int));

  if(p == NULL) {
  printf("dma is failed...\n");
  return 0;
  }

  printf("Enter the values\n");
  for(i=0;i<5;i++)
  scanf("%d",&p[i]);

  for(i=0;i<5;i++)
  printf("%d ",p[i]);
  printf("\n");
}
```



p i

0x1000 0

&p &i

Stack

p[0]	p[1]	p[2]	p[3]	p[4]
10	20	30	40	50
0x1000	0x1004	0x1008	0x1012	0x1016

p

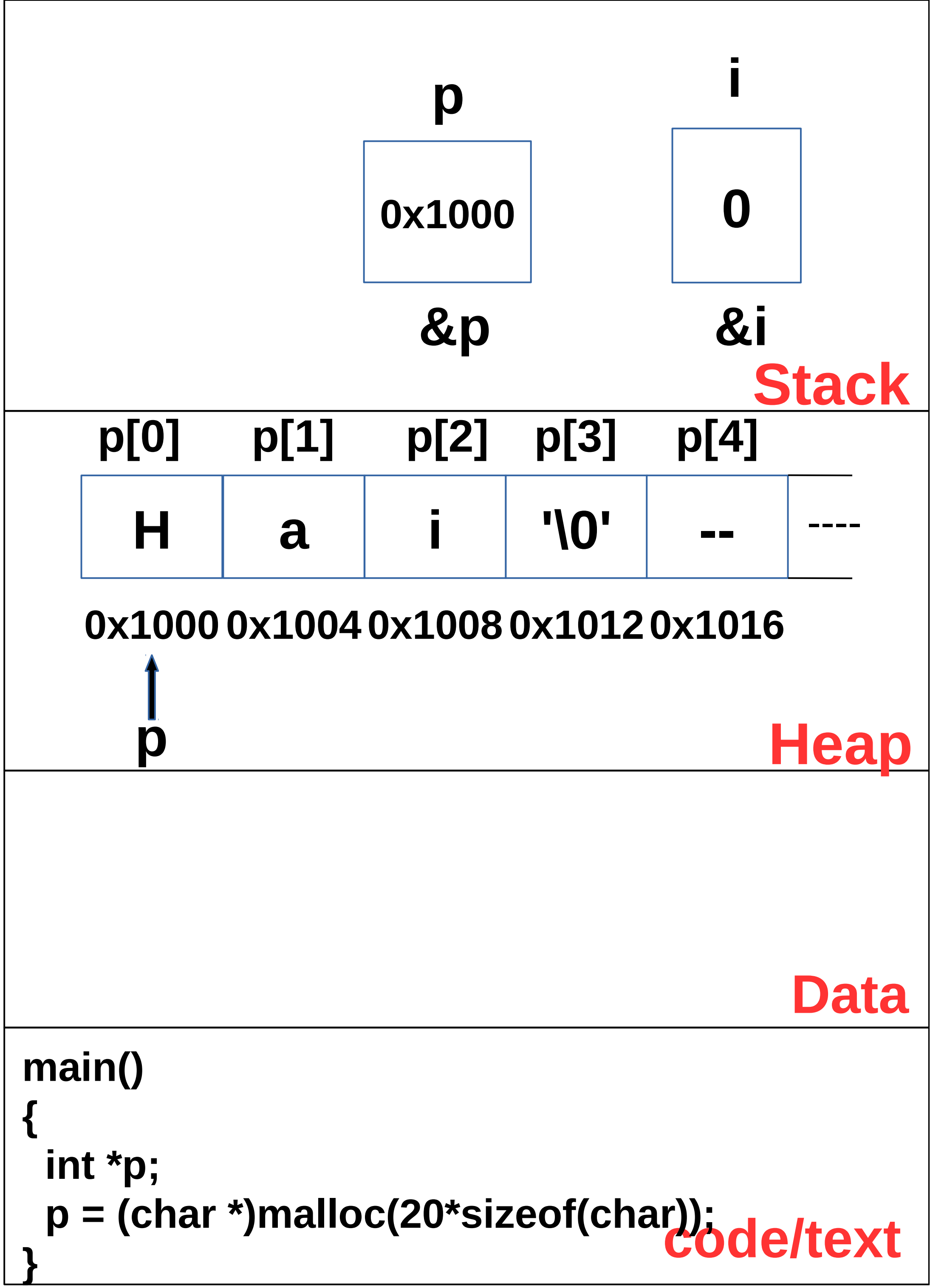
Heap

Data

```
main()
{
  int *p;
  p = (int *)malloc(5*sizeof(int));
}
```

code/text

```
#include<stdio.h>
main()
{
  char *p;
  p = (char *)malloc(20*sizeof(char));

  if(p == NULL) {
  printf(“dma is failed...\n”);
  return;
  }

  printf(“Enter ther string\n”);
  Scanf(“%s”,p);

  printf(“p = %s\n”,p);

}
```



p

0x1000

&p

i

0

&i

Stack

p[0]	p[1]	p[2]	p[3]	p[4]	
H	a	i	'\0'	--	----
0x1000	0x1004	0x1008	0x1012	0x1016	

p

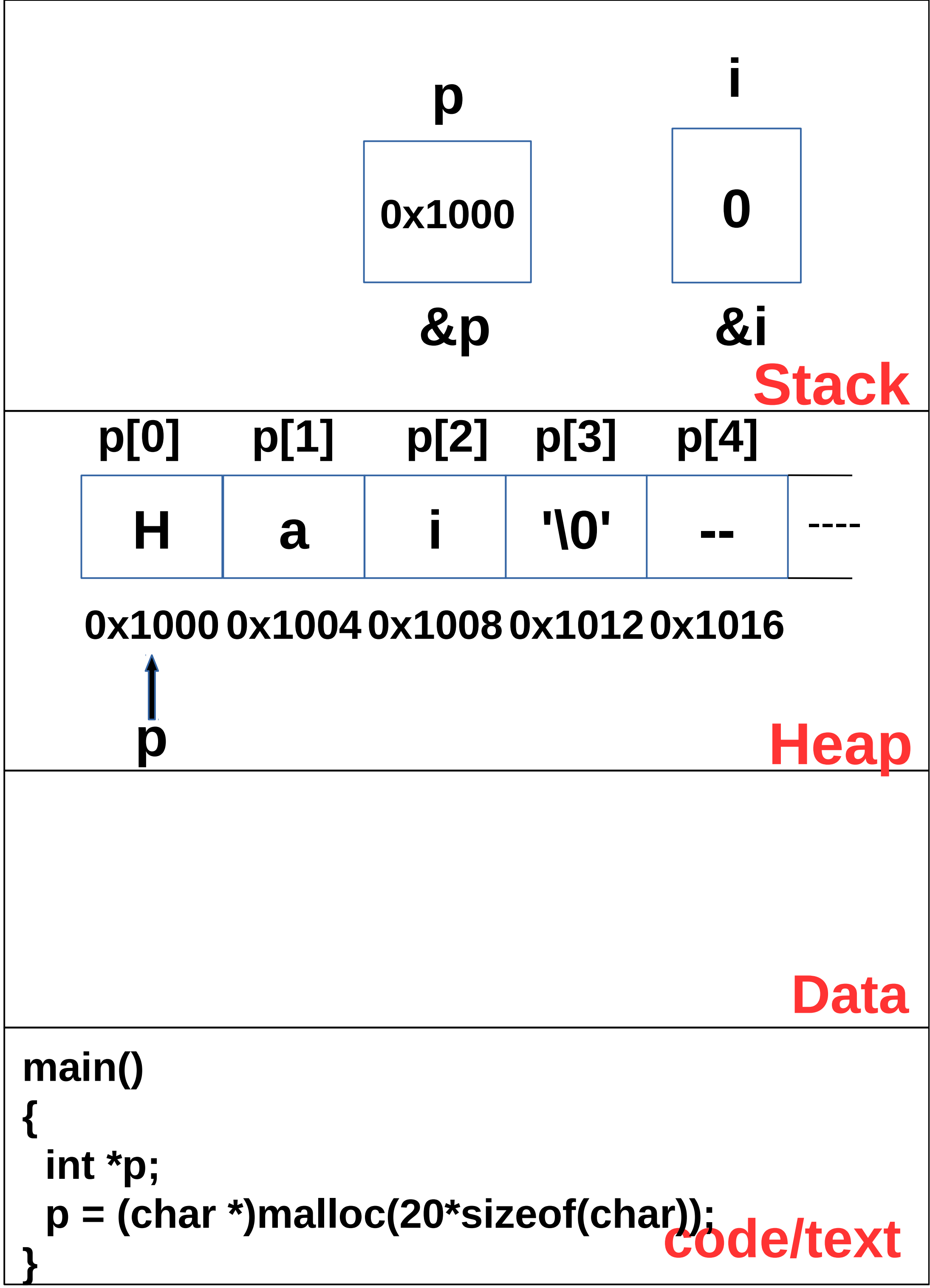
Heap

Data

```
main()
{
  int *p;
  p = (char *)malloc(20*sizeof(char));
}
```

code/text

```
#include<stdio.h>
main()
{
  char *p;
  p = (char *)malloc(20*sizeof(char));

  if(p == NULL) {
  printf(“dma is failed...\n”);
  return;
  }

  printf(“Enter the string\n”);
  Scanf(“%s”,p);

  printf(“p = %s\n”,p);

}
```



p

0x1000

&p

i

0

&i

Stack

p[0]	p[1]	p[2]	p[3]	p[4]	
H	a	i	'\0'	--	----
0x1000	0x1004	0x1008	0x1012	0x1016	

p

Heap

Data

```
main()
{
  int *p;
  p = (char *)malloc(20*sizeof(char));
}
```

code/text

SYNOPSIS

```
#include <stdlib.h>

void *calloc(size_t nmemb,size_t size);
```

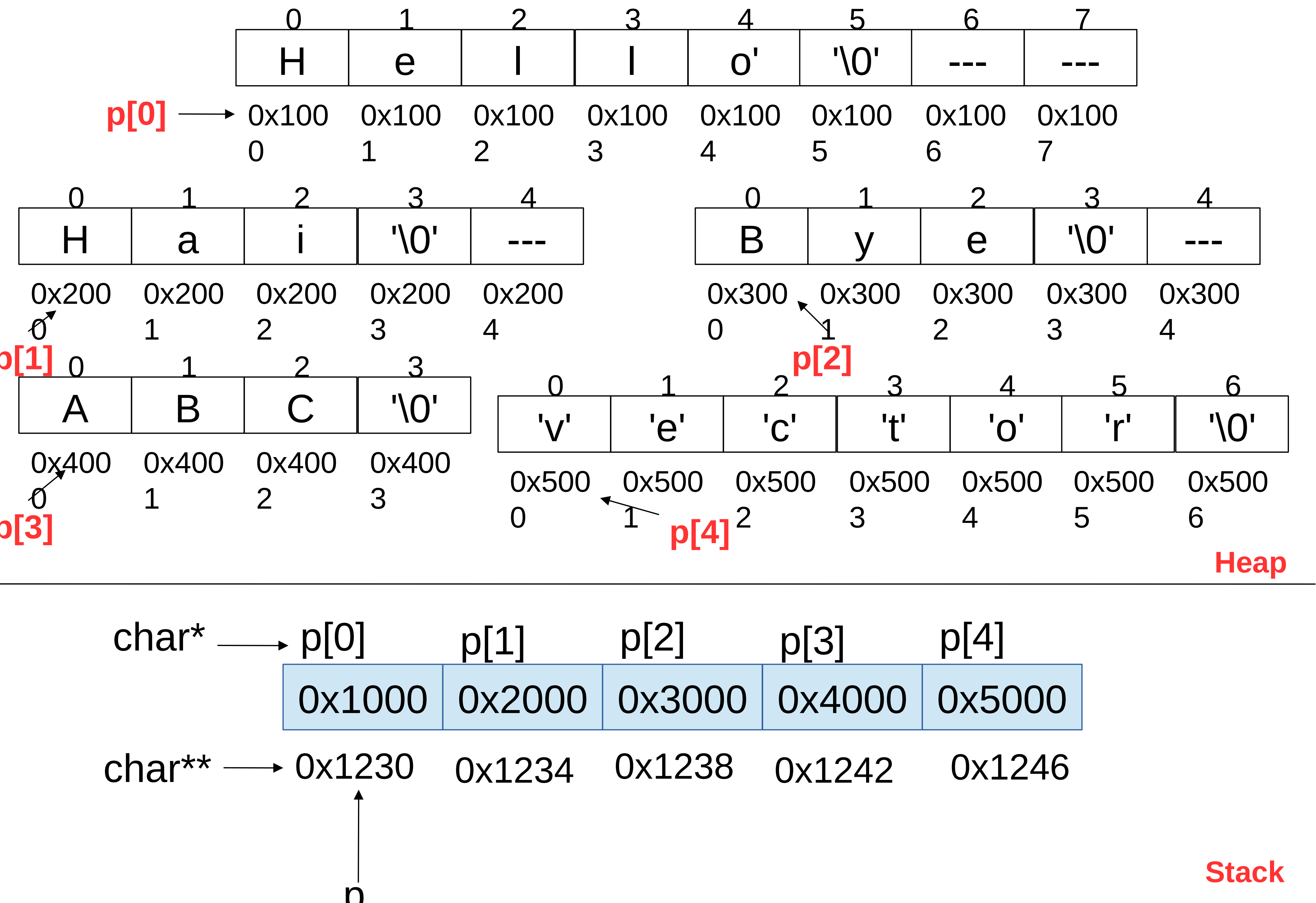
DESCRIPTION

The calloc() function allocates memory for an array of nmemb elements of size bytes each and returns a pointer to the allocated memory. The memory is set to zero. If nmemb or size is 0, then calloc() returns either NULL, or a unique pointer value that can later be successfully passed to free().

RETURN VALUES

Success : returns allocated memory base address.
Failure : returns NULL address.

Allocate memory for 5 strings



p[0] →

0	1	2	3	4	5	6	7
H	e	l	l	o'	'\0'	---	---
0x1000	0x1001	0x1002	0x1003	0x1004	0x1005	0x1006	0x1007

p[1]

0	1	2	3	4
H	a	i	'\0'	---
0x2000	0x2001	0x2002	0x2003	0x2004

p[2]

0	1	2	3	4
B	y	e	'\0'	---
0x3000	0x3001	0x3002	0x3003	0x3004

p[3]

0	1	2	3
A	B	C	'\0'
0x4000	0x4001	0x4002	0x4003

p[4]

0	1	2	3	4	5	6
'v'	'e'	'c'	't'	'o'	'r'	'\0'
0x5000	0x5001	0x5002	0x5003	0x5004	0x5005	0x5006

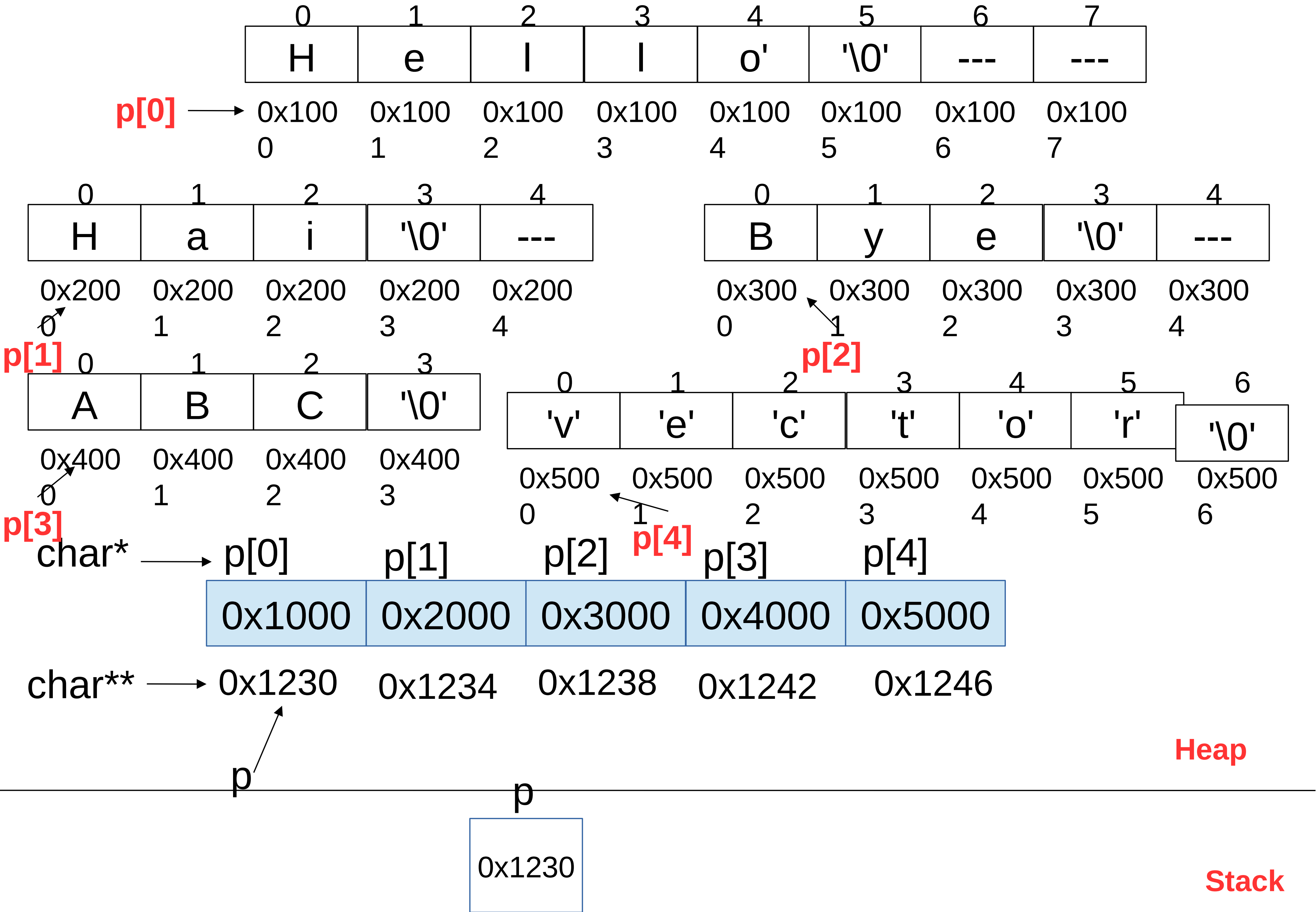
Heap

	p[0]	p[1]	p[2]	p[3]	p[4]
char* →	0x1000	0x2000	0x3000	0x4000	0x5000
char** →	0x1230	0x1234	0x1238	0x1242	0x1246

p

Stack

Allocate memory for n strings



p[0] →

0	1	2	3	4	5	6	7
H	e	l	l	o'	'\0'	---	---
0x100 0	0x100 1	0x100 2	0x100 3	0x100 4	0x100 5	0x100 6	0x100 7

p[1]

0	1	2	3	4
H	a	i	'\0'	---
0x200 0	0x200 1	0x200 2	0x200 3	0x200 4

p[2]

0	1	2	3	4
B	y	e	'\0'	---
0x300 0	0x300 1	0x300 2	0x300 3	0x300 4

p[3]

0	1	2	3
A	B	C	'\0'
0x400 0	0x400 1	0x400 2	0x400 3

p[4]

0	1	2	3	4	5	6
'v'	'e'	'c'	't'	'o'	'r'	'\0'
0x500 0	0x500 1	0x500 2	0x500 3	0x500 4	0x500 5	0x500 6

char* →

p[0]	p[1]	p[2]	p[3]	p[4]
0x1000	0x2000	0x3000	0x4000	0x5000

char** → 0x1230 0x1234 0x1238 0x1242 0x1246

p

Heap

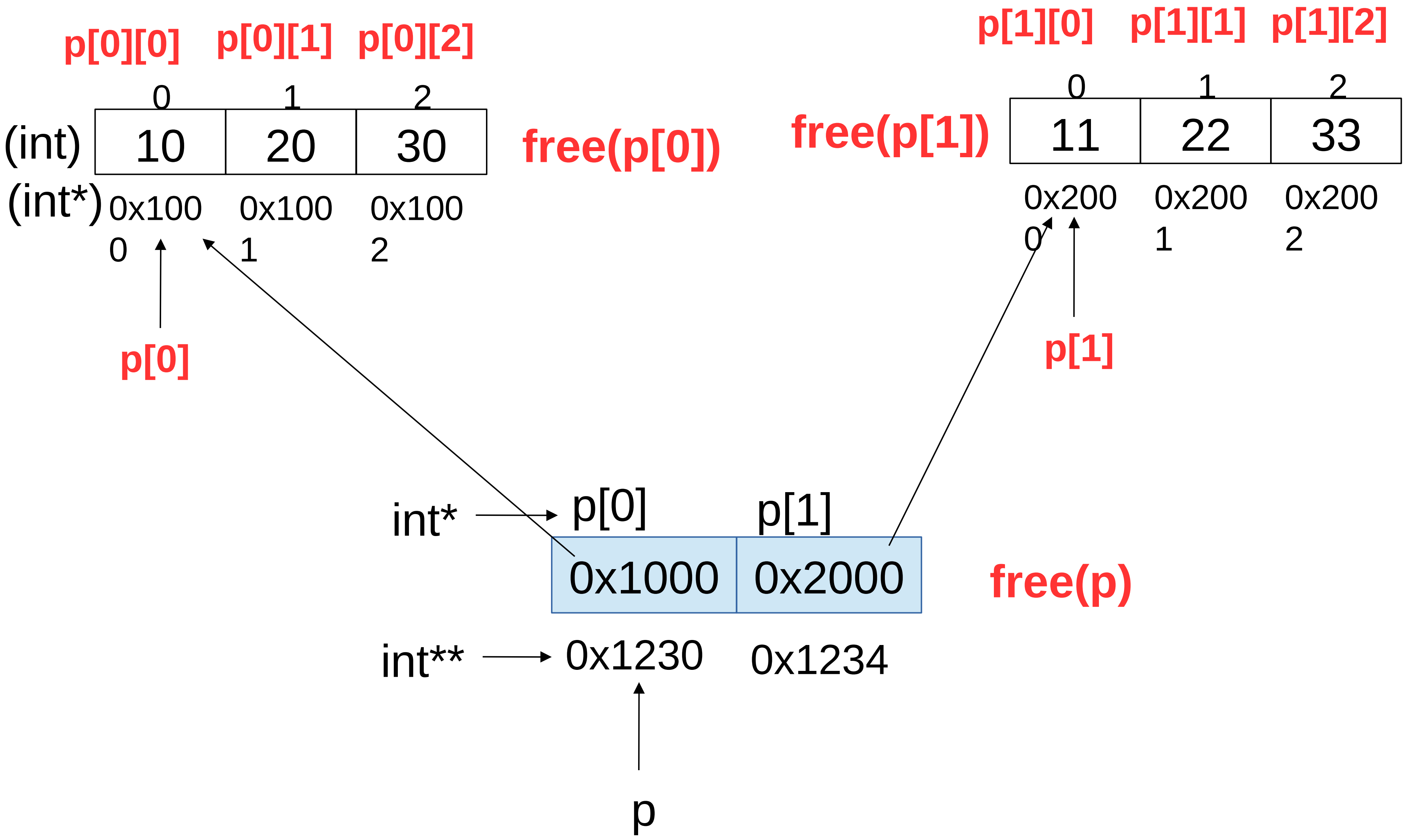
p

0x1230

Stack

```
//write a program to allocate a memory for n strings.
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int i,n;
        printf("Enter the number of strings\n");
        scanf("%d",&n);

        char **p = (char **)malloc(n * sizeof(char*)); //ary of ptr

        for(i=0;i<n;i++)
        p[i] = (char *)malloc(20*sizeof(char));

        printf("Enter the strings\n");
        for(i=0;i<n;i++)
        scanf("%s",p[i]);

        printf("display strings\n");
        for(i=0;i<n;i++)
        printf("%s\n",p[i]);

        for(i=0;i<n;i++)
        free(p[i]); //frees every strings memory

        free(p); //frees ary of ptr memory
}
```

Allocate memory for integer 2D array



```c
//write a program to allocate a memory for int 2D array
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int i,j,r,c;
        printf("Enter the number of rows & cols\n");
        scanf("%d%d",&r,&c);

         int **p = (int **)calloc(r,sizeof(int *)); //ary of ptr
         for(i=0;i<r;i++)
         p[i] = (int *)calloc(c,sizeof(int));

         printf("Enter the elements into 1D arrays\n");
         for(i=0;i<r;i++) {
         for(j=0;j<c;j++)
         scanf("%d",&p[i][j]);
         }

         printf("displaying the contents\n");
         for(i=0;i<r;i++) {
         for(j=0;j<c;j++)
         printf("%d  ",p[i][j]);
         printf("\n");
         }

         for(i=0;i<r;i++)
         free(p[i]); //free 1d
```

malloc()

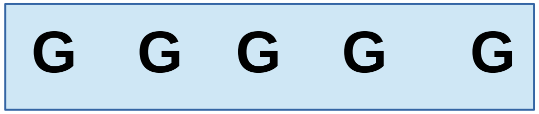
1. allocates memory as a single block and returns base address immediately

2. Default values are Garbage values.

3. faster in execution.

4. malloc takes only argument.

```
Ex : int *p;
       p = (int *)malloc(5*sizeof(int));
```



G G G G G

0x1000 20 bytes

calloc()

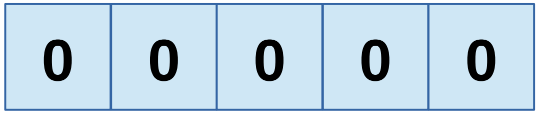
1. allocates memory as a multiple blocks and clears it and then returns base address.

2. Default values are 0's.

3. slower in execution.

4. calloc() takes 2 arguments.

```
Ex: int *p;
     p = (int *)calloc(5,sizeof(int));
```

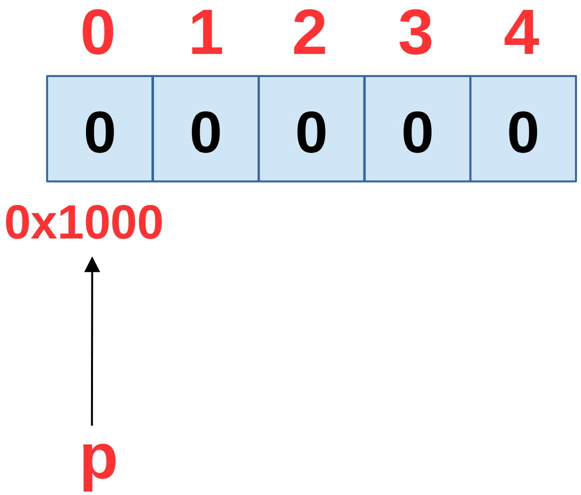


0 0 0 0 0

0x1000 20 bytes

```
#include<stdlib.h>
int main()
{
        int *p,i;
        p = (int *)malloc(5*sizeof(int));

        printf("p = %p\n",p);
        for(i=0;i<5;i++)
        printf("%d ",p[i]);
        printf("\n");

        for(i=0;i<5;i++)
        p[i] = i+10;

        for(i=0;i<5;i++)
        printf("%d ",p[i]);
        printf("\n");

        free(p);

        printf("p = %p\n",p);
        for(i=0;i<5;i++)
        printf("%d ",p[i]);
        printf("\n");
}
```

```
p = (int *)malloc(5*sizeof(int));
```

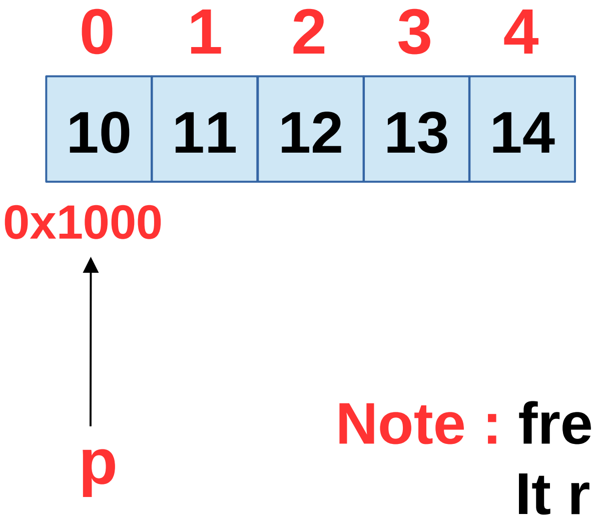


0	1	2	3	4
0	0	0	0	0

0x1000

p

```
for(i=0;i<5;i++)
p[i] = i+10;
```

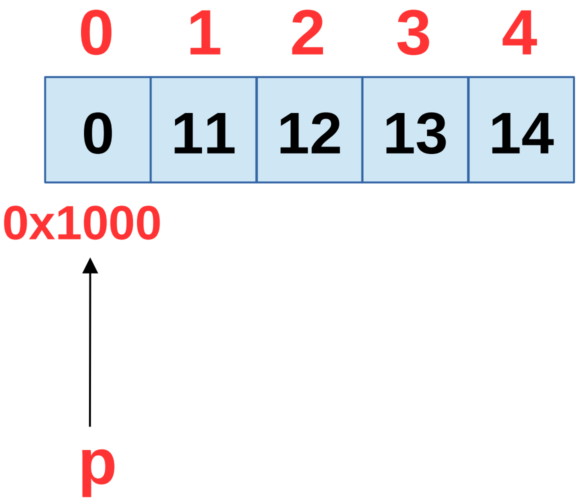


0	1	2	3	4
10	11	12	13	14

0x1000

p

```
free(p);
```



0	1	2	3	4
0	11	12	13	14

0x1000

p

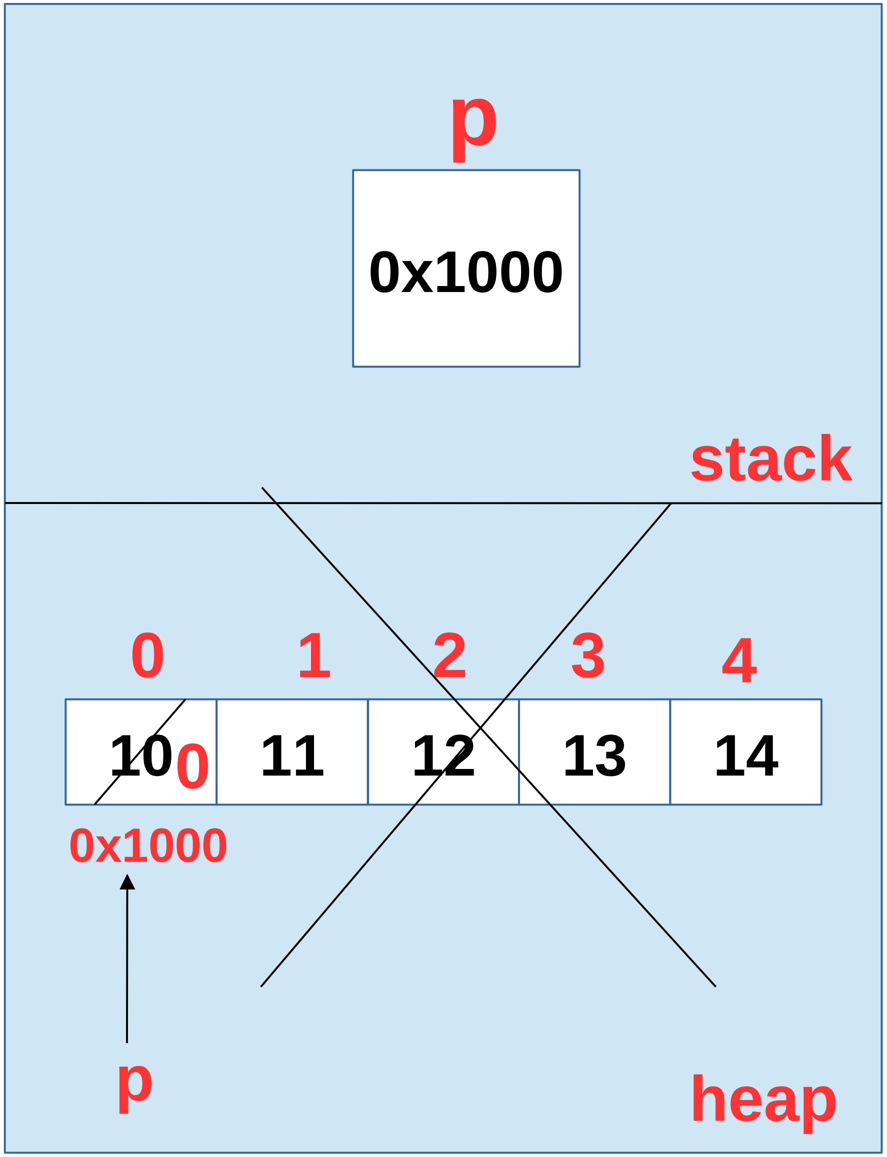
```
void free(void *p)
{
     //logic to free the memory.

     *p = 0;
}
```

Note : freeing memory means not clearing the memory.
It removes only reservation.

--> clearing memory means put 0's in memory.

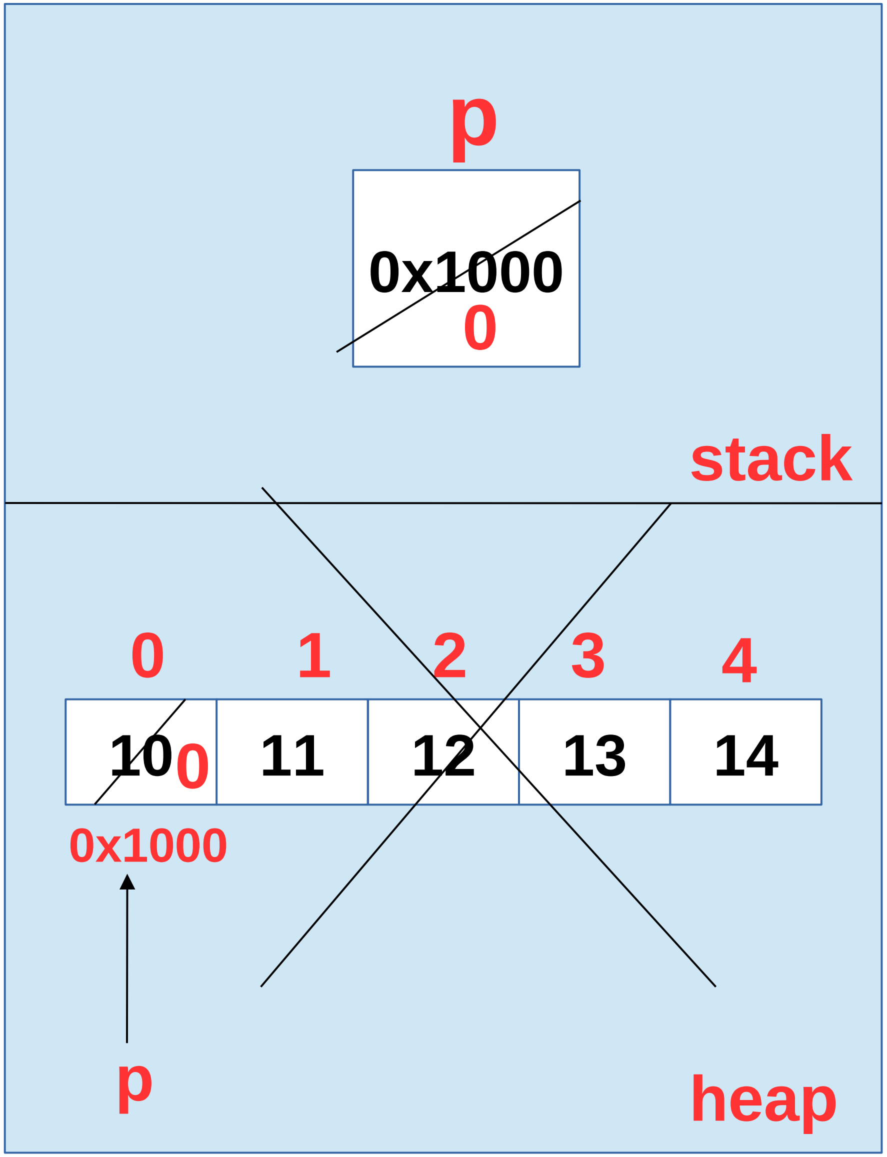
Dangling pointer : **Even after freeing the memory also, still pointer Points to same memory location is called dangling pointer.**

free(p)

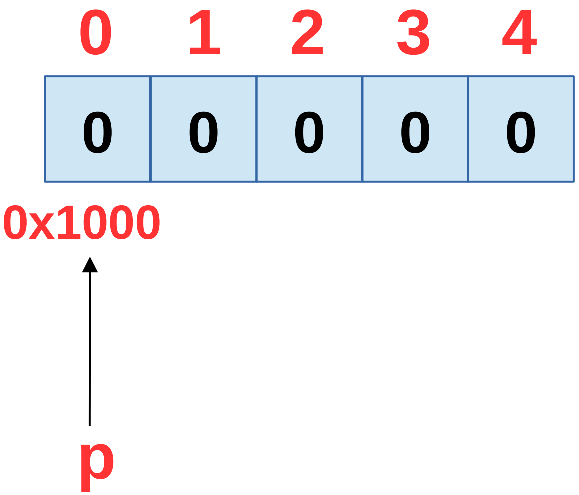


How to avoid Dangling pointer?

```
free(p);
p = NULL;
```



```
p = (int *)malloc(5*sizeof(int));
```

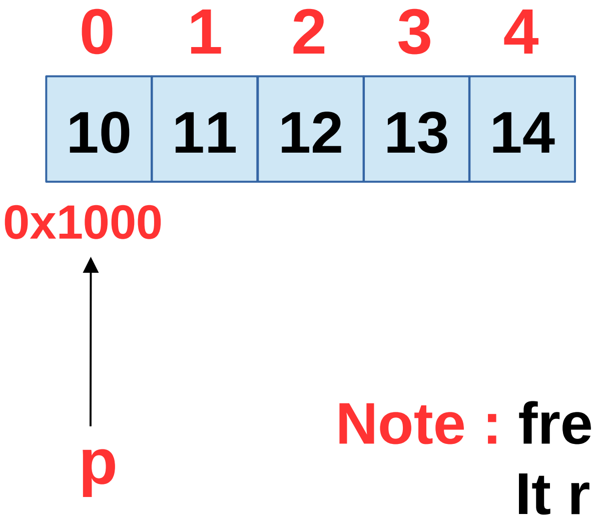


0	1	2	3	4
0	0	0	0	0

0x1000

p

```
for(i=0;i<5;i++)
p[i] = i+10;
```

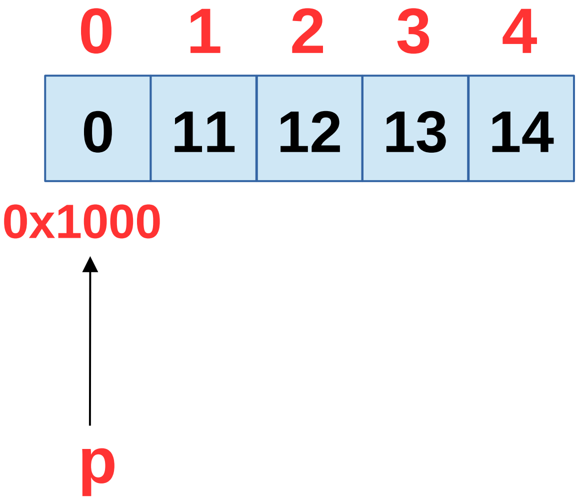


0	1	2	3	4
10	11	12	13	14

0x1000

p

```
free(p);
```



0	1	2	3	4
0	11	12	13	14

0x1000

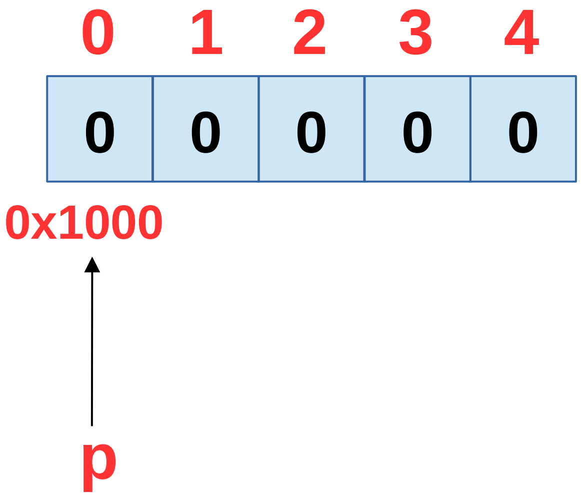
p

```
void free(void *p)
{
      //logic to free the memory.

      *p = 0;
}
```

Note : **freeing memory means not clearing the memory.**
It removes only reservation.

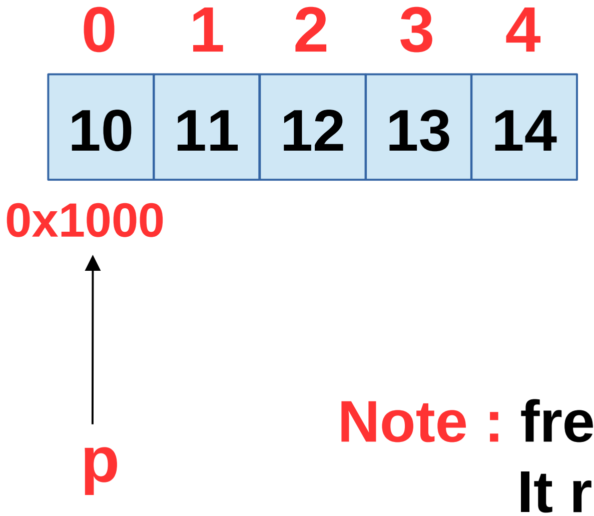
--> clearing memory means put 0's in memory.

```
p = (int *)malloc(5*sizeof(int));
```



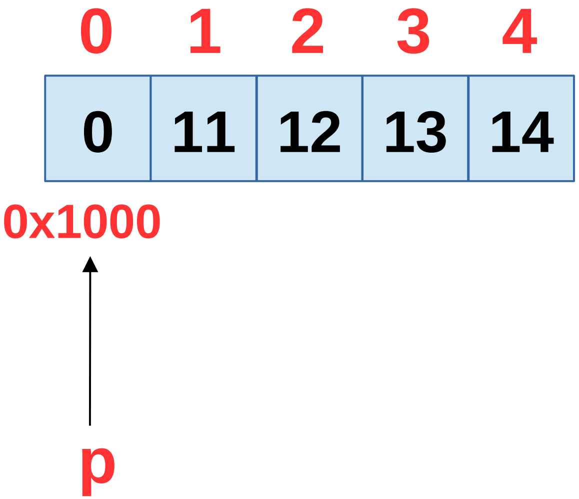
0	1	2	3	4
0	0	0	0	0

```
for(i=0;i<5;i++)
p[i] = i+10;
```



0	1	2	3	4
10	11	12	13	14

```
free(p);
```



0	1	2	3	4
0	11	12	13	14

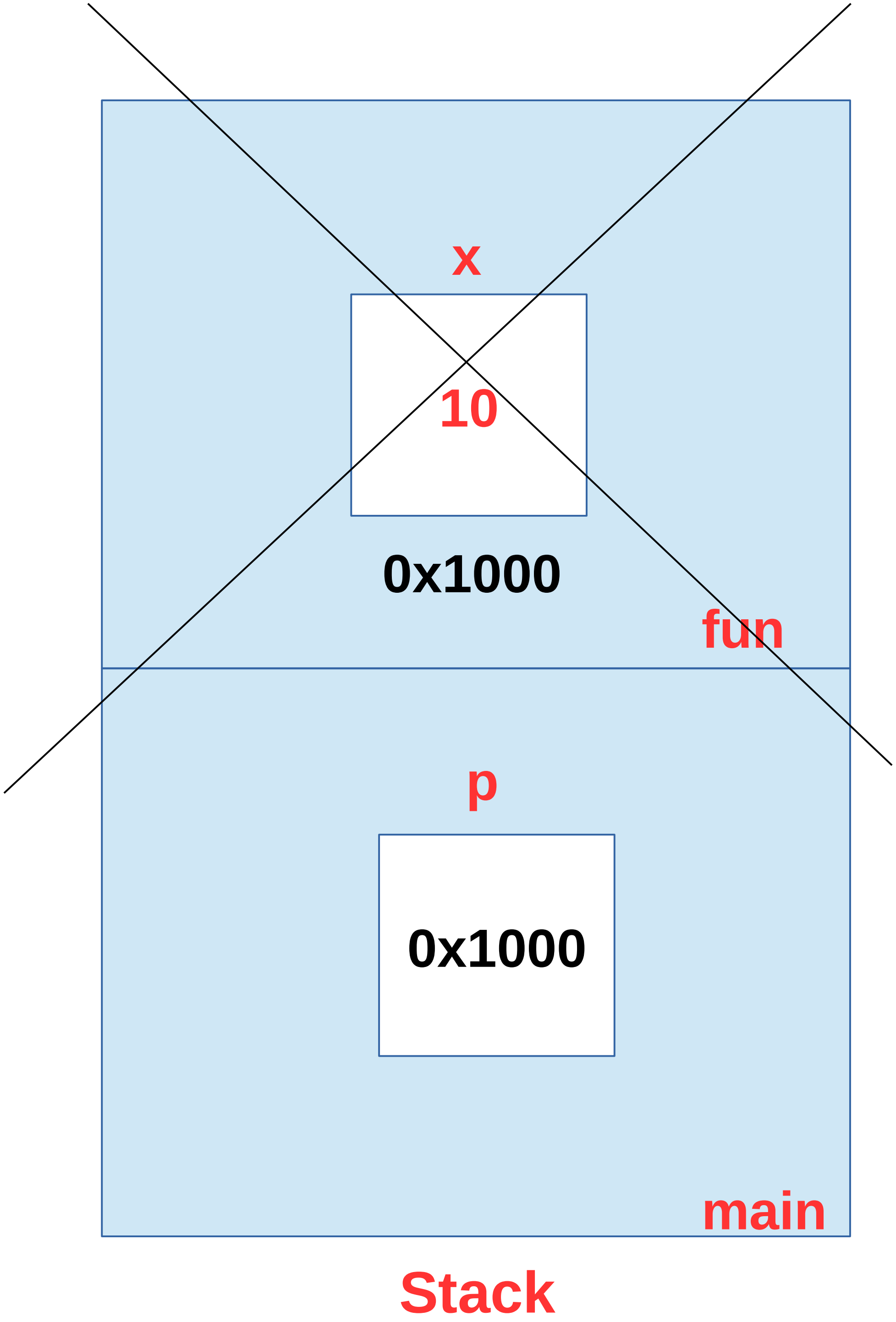
```
void free(void *p)
{
      //logic to free the memory.

      *p = 0;
}
```

Note : freeing memory means not clearing the memory.
It removes only reservation.

--> clearing memory means put 0's in memory.

```
#include<stdio.h>
int* fun();
int main()
{
        int *p = fun();
        printf("in main(), p = %p\n",p);
        printf("*p = %d\n",*p);
}
int* fun()
{
        int x = 10;
        printf("in fun(), &x = %p\n",&x);
        return &x;
}
```

//Note : in main(), p is called as
dangling pointer.



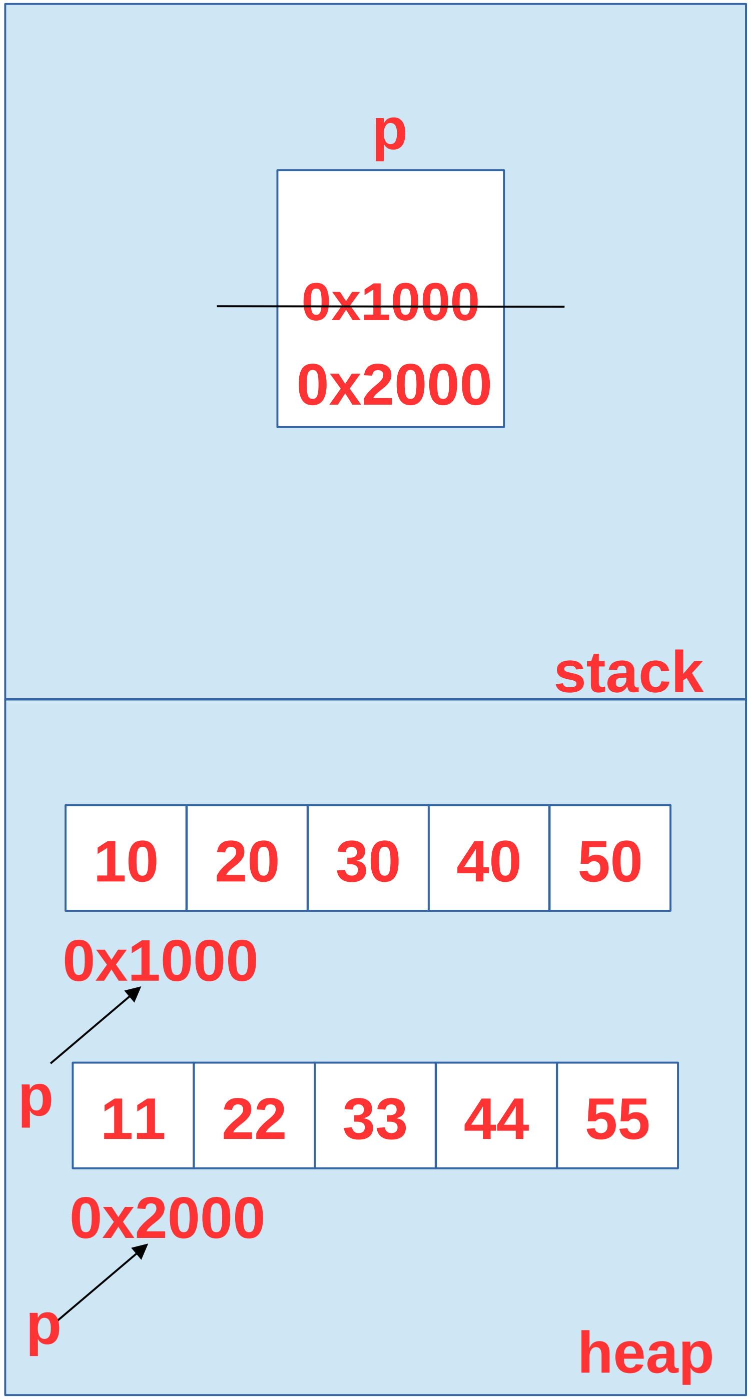
Memory leak

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *p = (int *)malloc(5*sizeof(int));
        printf("p = %p\n",p);

        p = (int *)malloc(5*sizeof(int));
        printf("p = %p\n",p);
}
```

Note : 0x1000 address memory is leaked.

Memory leak : unused memory bytes.

In DMA, if a pointer points to another memory, with out freeing the old memory, then it is called as memory leak.

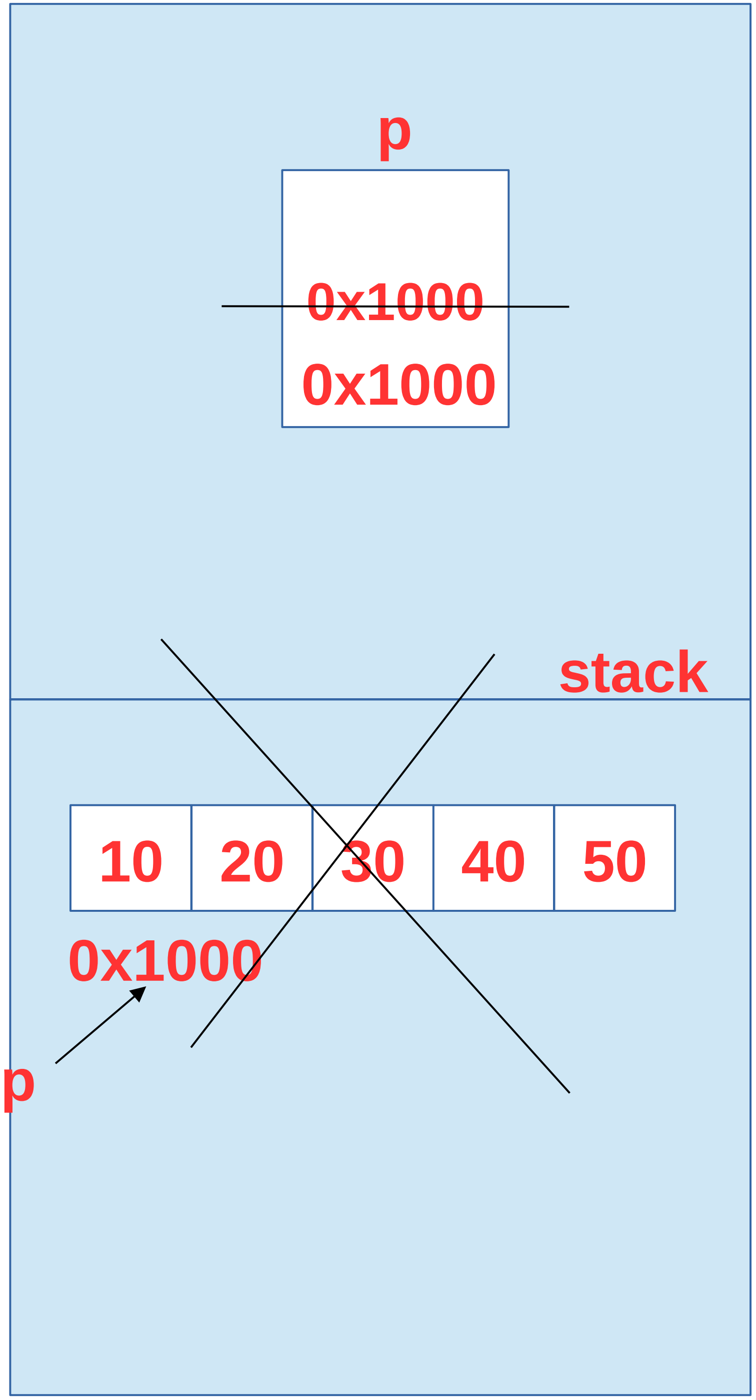


No Memory leak

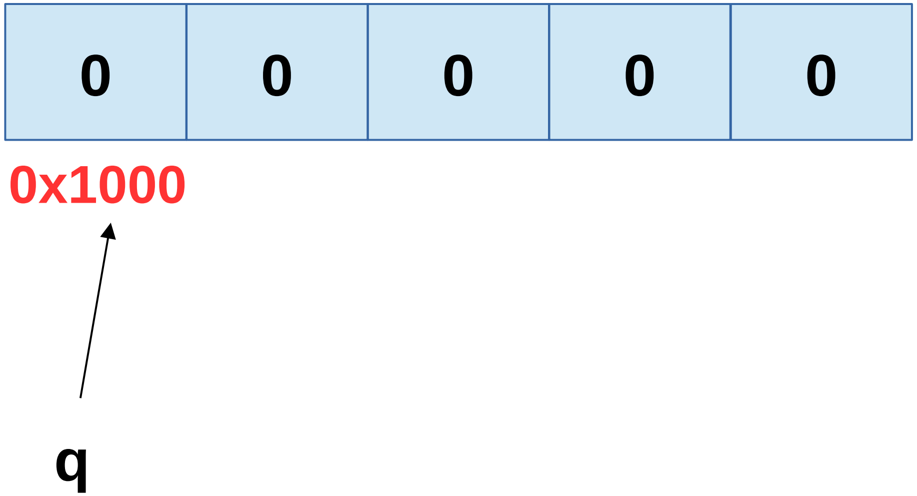
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *p = (int *)malloc(5*sizeof(int));
        printf("p = %p\n",p);

        free(p);

        p = (int *)malloc(5*sizeof(int));
        printf("p = %p\n",p);
}
```

Q. How to avoid memory leak in DMA?

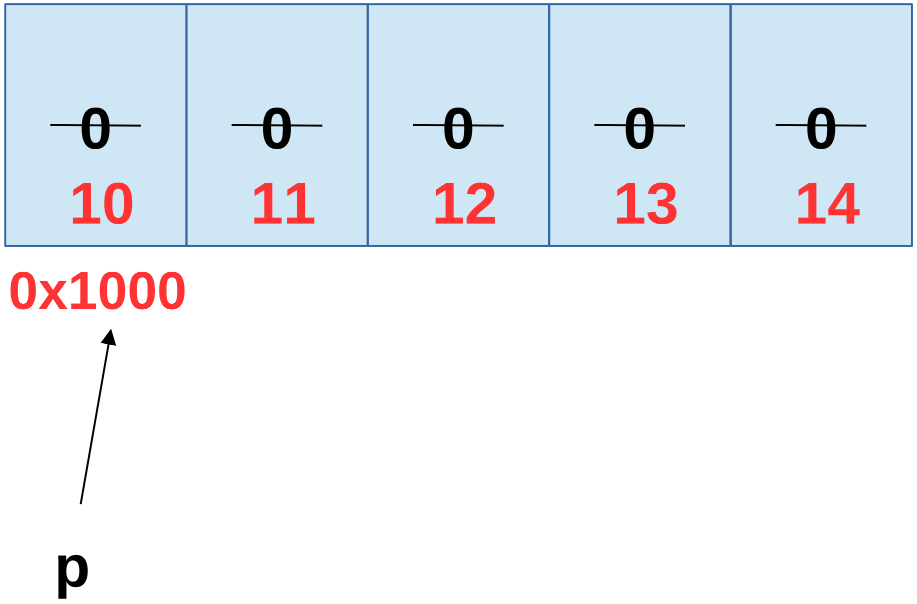
A. using free().



```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *q,i;
        q = (int *)malloc(5*sizeof(int));

        printf("q = %p\n",q);

        for(i=0;i<5;i++)
        printf("%d  ",q[i]);
        printf("\n");
}
```



```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *p,*q,i;

        p = (int *)malloc(5*sizeof(int));
        printf("p = %p\n",p);

        for(i=0;i<5;i++)
        p[i] = i+10;

        free(p);
}
```



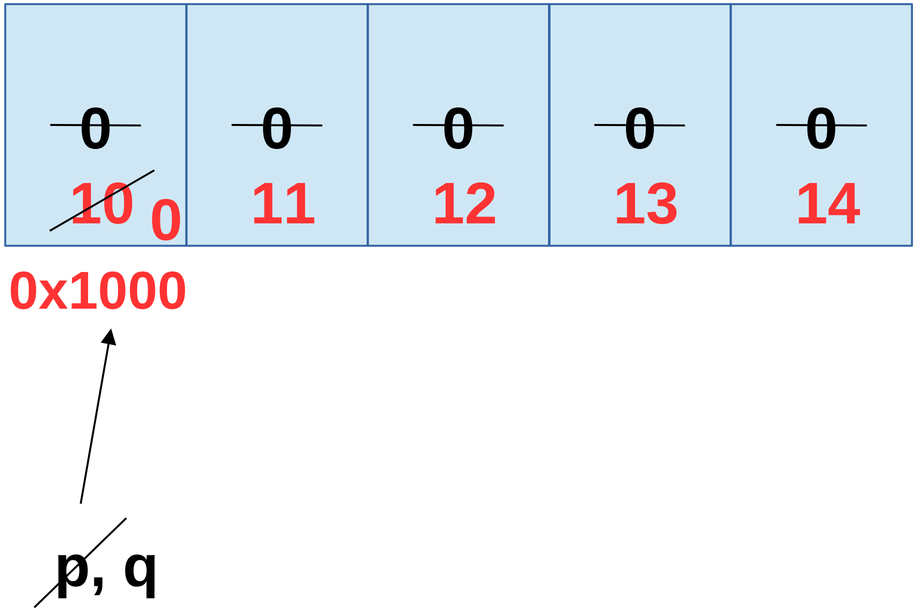
~~0~~ ~~0~~ ~~0~~ ~~0~~ ~~0~~

10 11 12 13 14

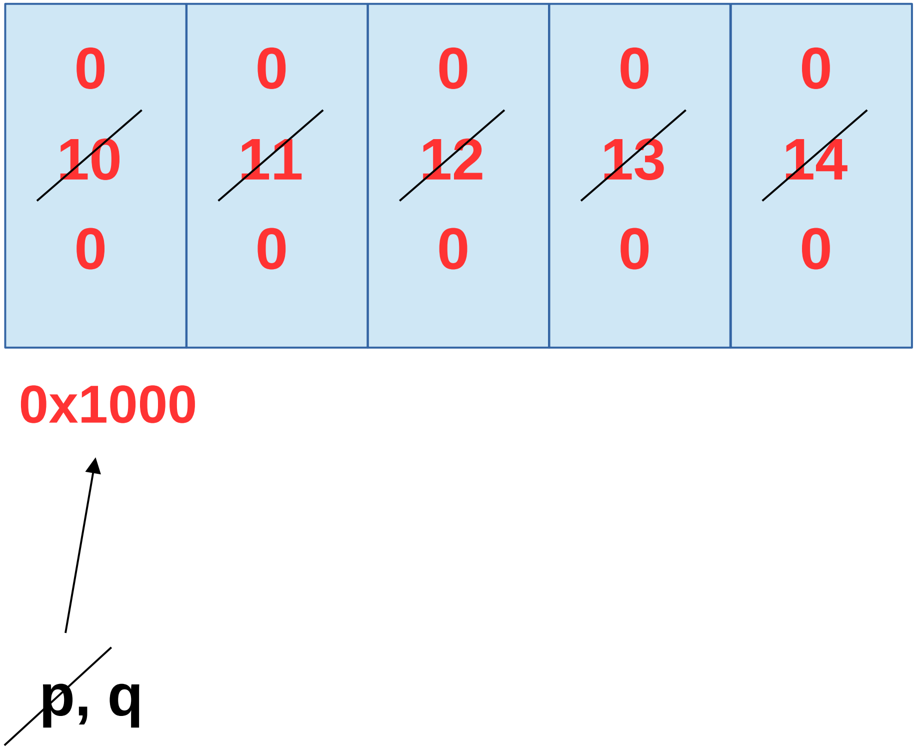
0x1000

p

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *p,*q,i;

        p = (int *)malloc(5*sizeof(int));
        printf("p = %p\n",p);

        for(i=0;i<5;i++)
        p[i] = i+10;

        free(p);

        q = (int *)malloc(5*sizeof(int));
        printf("q = %p\n",q);

        for(i=0;i<5;i++)
        printf("%d  ",q[i]);
        printf("\n");
}
```



```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *p,*q,i;

        p = (int *)malloc(5*sizeof(int));
        printf("p = %p\n",p);

        for(i=0;i<5;i++)
        p[i] = i+10;

        free(p);
        p = NULL;

        q = (int *)calloc(5*sizeof(int));
        printf("q = %p\n",q);

        for(i=0;i<5;i++)
        printf("%d  ",q[i]);
        printf("\n");
}
```



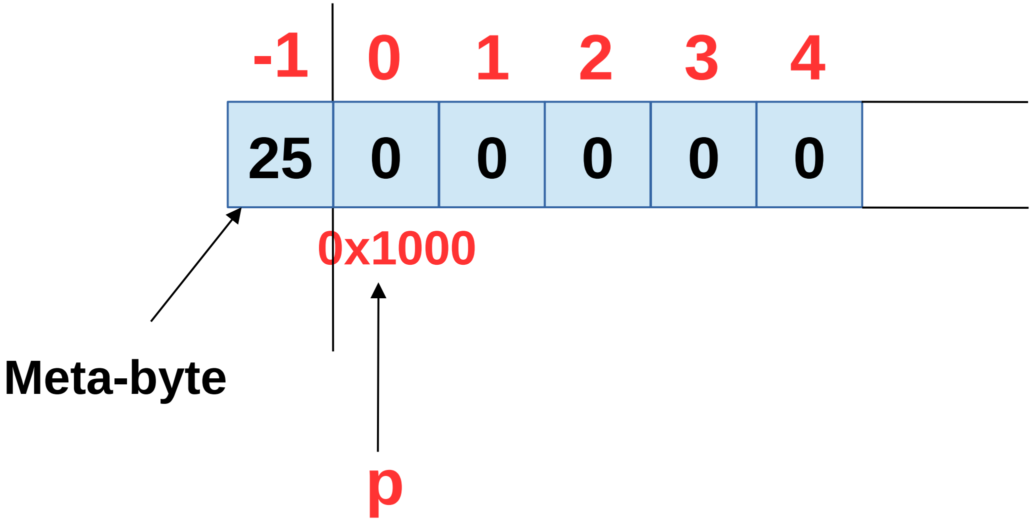
0 ~~10~~ 0	0 ~~11~~ 0	0 ~~12~~ 0	0 ~~13~~ 0	0 ~~14~~ 0

0x1000

~~p~~, q

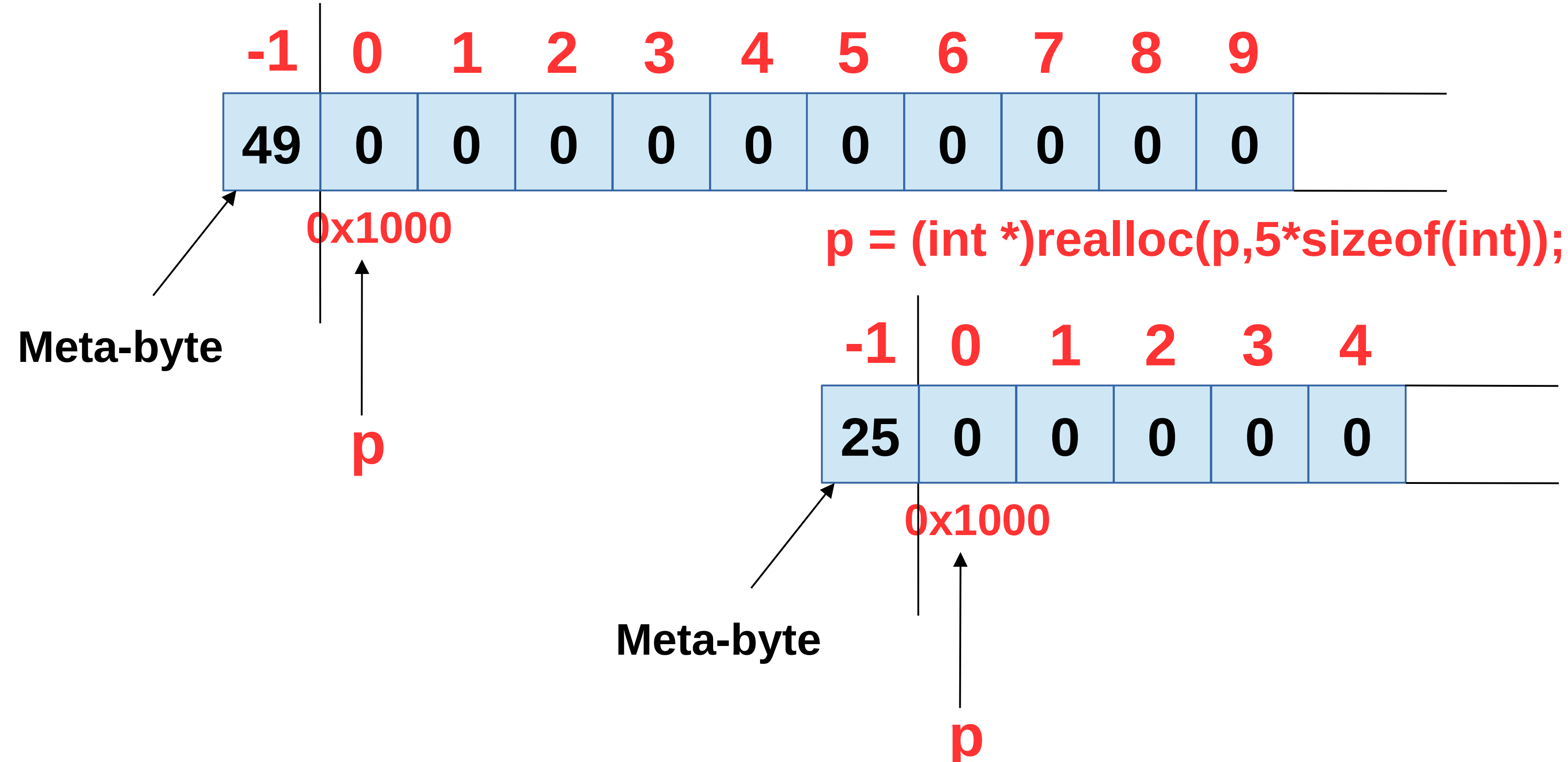
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *p = (int *)malloc(5*sizeof(int));
        printf("p = %p\n",p);
        printf("no.of bytes : %d\n",p[-1]);

        p = (int *)realloc(p,10*sizeof(int));
        printf("p = %p\n",p);
        printf("no.of bytes : %d\n",p[-1]);

        p = (int *)realloc(p,5*sizeof(int));
        printf("p = %p\n",p);
        printf("no.of bytes : %d\n",p[-1]);
}
```

```
p = (int *)malloc(5*sizeof(int));
```



-1	0	1	2	3	4
25	0	0	0	0	0

```
p = (int *)realloc(p, 10*sizeof(int));
```



-1	0	1	2	3	4	5	6	7	8	9
49	0	0	0	0	0	0	0	0	0	0

```
p = (int *)realloc(p,5*sizeof(int));
```

-1	0	1	2	3	4
25	0	0	0	0	0

```
#include<stdio.h>
#include<malloc.h>
int main()
{
	int *p = (int *)realloc(0,5*sizeof(int));
//above statement equals to  p = (int *)malloc(5*sizeof(int));

	printf("p = %p\n",p);
	printf("no.of bytes = %d\n",p[-1]);

	p = (int *)realloc(p,0);
	printf("p = %p\n",p);
	//printf("no.of bytes = %d\n",p[-1]);
}
```