# Pointers

# Characters and strings

- **Strings**
  - **Definition: A sequence of characters** are treated as a single unit, called as string.
    - Can include letters, digits and special characters (*, /, $)
  - **String literal** (string constant) - written in double quotes
    - **"Hello"**
  - **Strings are arrays of characters**
    - String - a pointer to first character
    - Value of string - is the address of first character

# Memory Storage for a String

- The string is always ended with a null character '\0'.
- The characters after the null character are ignored.
- **Example:**

```
char str[20] = "Initial value";
```

# Strings - Declarations

- **String declarations**
  - Declare as a character array or a variable of type **char ***

    **char name[] = "ram";**

    **char *nameptr = "ram";**
  - Remember that strings represented as character arrays end with **'\0' (A NULL Character)**
- **How to read string as input?**
  - Use **scanf** function

    **Example: scanf("%s", name);**
    - Copies input into **name[]**
    - Do not need **& (because a string is a pointer)**
  - We must give one character of array for **'\0'**

# Strings

```
#include <stdio.h>
int main ()
{
char name[6] = {'A', 'C', '\0', 'T', 'S', '\0'};
printf("message: %s\n", name );
return 0;
}
```

# Strings – Reading and writing

```
#include <stdio.h>
int main()
{
        char str1[5] = "desd";
        char *str2="ram07";
        char str3[5];
        printf("size of str1 = %d\n", sizeof(str1)); //String size
or pointer size ???
        printf("size of str2 = %d\n", sizeof(str2)); //pointer
size or pointer size ???
            return 0;
}
```

# Strings – Reading and writing

```
#include <stdio.h>
int
main ()
{
  char str1[5];
  char *str2;
    printf ("enter a string....\n");
  scanf ("%s", str1);
  printf ("entered  string..str1..is %s \n", str1);
  printf ("enter a string....\n");
  scanf ("%s", str2); // error or warning or segmentation fault
  printf ("entered  string..str2..is %s \n", str2);
  return 0;
}
```

# All Pointers

```
#include <stdio.h>
void main()
{
    char ch = 'c';
    char *chptr = &ch;

    int i = 20;
    int *intptr = &i;

    float f = 1.20000;
    float *fptr = &f;

    char *ptr = “C Program";
    printf("\n [%c], [%d], [%f], [%c], [%s]\n", *chptr, *intptr, *fptr,
*ptr, ptr);
}
```

# Pointers

char *ptr = “C Program";


**the last character, its a null character which is placed at the end of every string by default**

printf(“ [%c]", *ptr);

any character pointer pointing to a string stores the address of the first character of the string.

‘ptr’ holds the address of the character ‘C’ ie 1000.

‘value of’ operator ‘*’ to ‘ptr’, fetch the value at address 1000 which is ‘C’

when we print ‘*ptr’,  ‘C’ gets prints printed

# Pointers

char *ptr = “C Program";


specify the format specifier as ‘%s’ and use ‘ptr’ (which contains the starting address of the string), then the complete string is printed using printf. The concept is that %s specifier requires the address of the beginning byte of string to display the complete string, which we provided using ‘ptr’ (which we know holds the beginning byte address of the string).

C Program

# Strings – library functions

**strcpy(s1, s2)**
Copies string s2 into string s1.

**strcat(s1, s2);**
Concatenates string s2 onto the end of string s1.

**strlen(s1);**
Returns the length of string s1.

**strcmp(s1, s2)**
Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.

# Strings – library functions

**strchr(s1, ch)**
Returns a pointer to the first occurrence of character ch in string s1.

**strstr(s1, s2);**
Returns a pointer to the first occurrence of string s2 in string s1.

# String length

```
#include<stdio.h>
int length(char* );
void main(void)
{
   char str1[50];
   int len;
   printf("\nEnter String whose
length has to be found:");

scanf("%s",str1);

 len=length(str1);

 printf("\nlength of String %s is
%d",str1,len);
}
```

```
int length(char *s1)
{
    int l=0;
    while(*s1!='\0')
    {
        l++;
        s1++;
    }

     return l;
}
```

# String concatenation

```
#include<stdio.h>
void concat(char* ,char* );
void main(void)
{
   char str1[25],str2[25];
   printf("\nEnter First String:");
scanf("%s",str1);

   printf("\nEnter Second String:");
scanf("%s",str2);

   concat(str1,str2);
   printf("\nConcatenated String is
%s",str1);
}
```

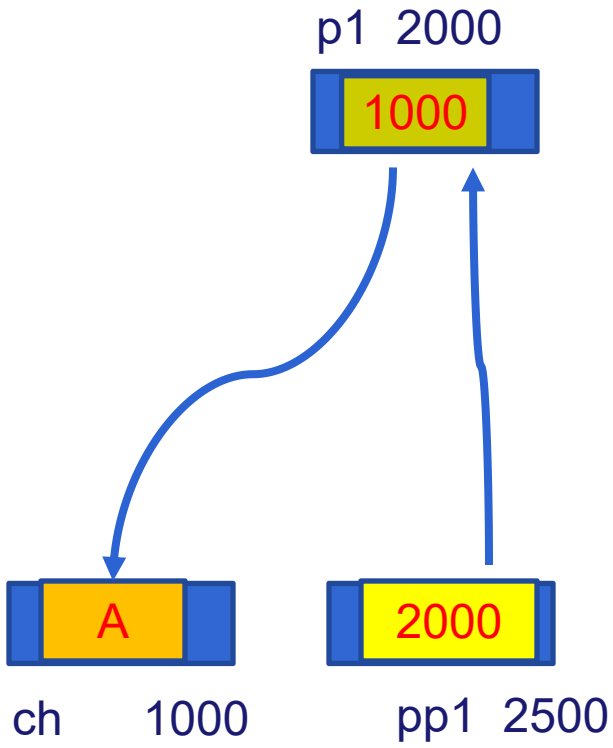
```
void concat(char *s1,char *s2)
{
    while(*s1!='\0')
          s1++;
    while(*s2!='\0')
    {
          *s1=*s2;
          s1++;
          s2++;
    }
     *s1='\0';
}
```

# Pointer to Pointer

# Pointer to Pointer

As the definition of pointer says that its a special variable that can store the address of an other variable. Then the other variable can very well be a pointer. This means that its perfectly legal for a pointer to be pointing to another pointer

Lets suppose we have a pointer ‘pp1′ that points to yet another pointer ‘p1′ that points to a character ‘ch’.



# Pointer to Pointer

in memory, pointer pp1 holds the address of pointer p1. Pointer p1 holds the address of character ‘ch’.

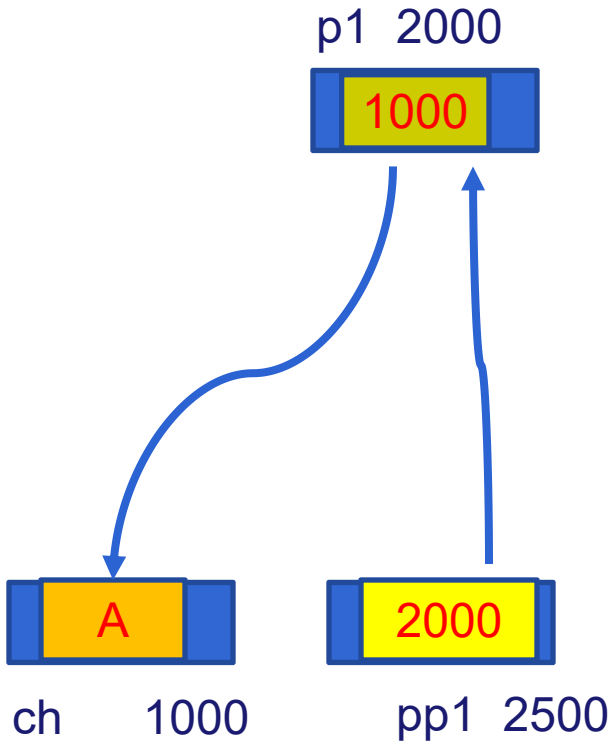
So ‘p1′ is pointer to character ‘A’, while ‘pp1′ is pointer to ‘p1′ or we can also say that ‘pp1′ is a pointer to pointer to character ‘ch’.

Now,

‘pp1′ is the address of ‘p1′ ie 2000

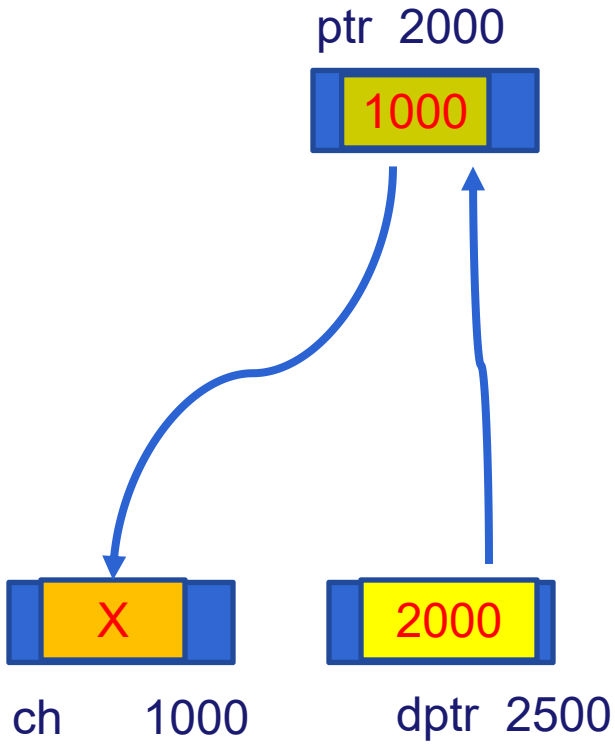
‘*pp1′ is the value held by ‘p1′ ie 1000

‘**p1′ is the value at 1000 ie ‘A’



# Pointer to Pointer Example:

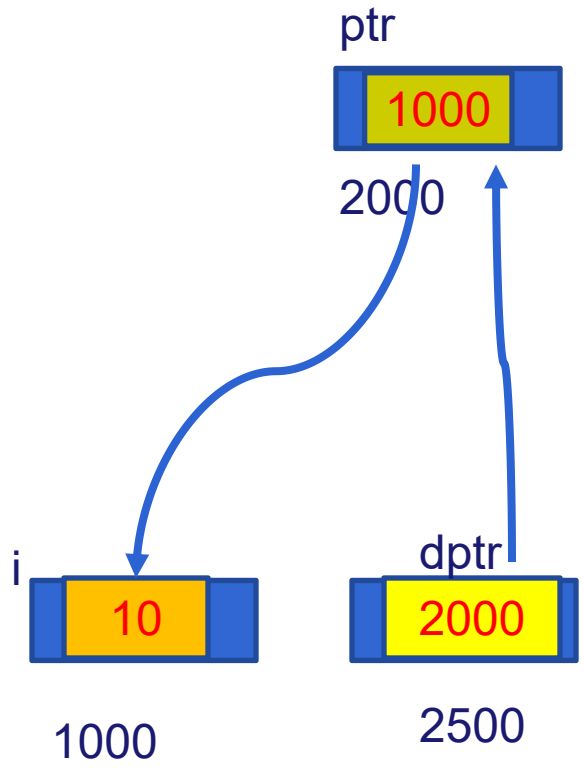
```
#include<stdio.h>
int
main (void)
{
  char ch = 'A';
  char *ptr = NULL;
  char **dptr = NULL;
  ptr = &ch;
  dptr = &ptr;
  printf ("\n ch = [%c]\n", ch);
  printf ("\n *ptr = [%c]\n", *ptr);
  printf ("\n **dptr = [%c]\n", **dptr);
  return 0;
}
```




**What is the value of ch ,*ptr and **dptr?**

# Pointer to Pointer Example:

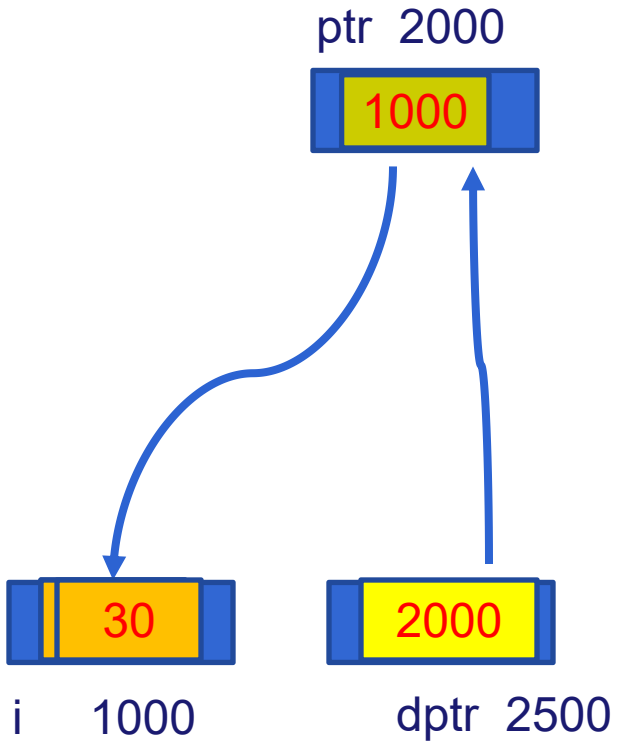
```
#include<stdio.h>
int
main (void)
{
  int  i= 10;
  int *ptr = NULL;
  int **dptr = NULL;
  ptr = &i;
  dptr = &ptr;
  printf ("\n i = [%d]\n", i);
  printf ("\n *ptr = [%d]\n", *ptr);
  printf ("\n **dptr = [%d]\n", **dptr);
  return 0;
}
```




**What is the value of dptr and &ptr?**

# Pointer to Pointer Example:

```
#include<stdio.h>
int
main (void)
{
   int i = 10;
  int *ptr = NULL;
  int **dptr = NULL;
  ptr = &i;
  dptr = &ptr;
  printf ("\n i = [%d]\n", i);
  *ptr = 20;
  printf ("\n i = [%d]\n", i);
  **dptr = 30;
  printf ("\n i = [%d]\n", i);
  return 0;
}
```



# 1D – array

```
#include <stdio.h>
int main()
{
        int num[5] = {21,18,57,45,50};
        int i;
        for(i=0;i<5;i++)
        {
         printf("\n  %d element addr: %p",i,num+i);
         printf("\n Integer Array Element *(num + %d) : %d  ",i,*(num+i));
         }
                return 0;
 }
```

# 1D – array

```
#include <stdio.h>
int main()
{
	int num[5] ;
	int i;
	for(i=0;i<5;i++)
	{
		printf("\n  %d element addr: %p",i,num+i);
		 printf("\n Integer Array Element num[%d] : %d  ",i,num[i]);	}

	for(i=0;i<5;i++)
	{
		printf("\n  enter %d element ",i);
		scanf("%d",&num[i]);
	}

	for(i=0;i<5;i++)
	{
		printf("\n Integer Array Element num[%d] : %d  ",i,num[i]);
	}

return 0;
}
```

# 1D – array

```
#include <stdio.h>
int main()
{
	int num[5] ;
	int i;
	for(i=0;i<5;i++)
	{
		printf("\n  %d element addr: %p",i,num+i);
		printf("\n Integer Array Element *(num + %d) : %d  ",i,*(num+i));
	}

	for(i=0;i<5;i++)
	{
		printf("\n  enter %d element ",i);
		scanf("%d",num+i);
	}

	for(i=0;i<5;i++)
	{
		printf("\n Integer Array Element *(num + %d) : %d  ",i,*(num+i));
	}

return 0;
}
```

# Multidimensional Arrays

**Representing a double dimension Array**

*integer a[3] [4] ;*

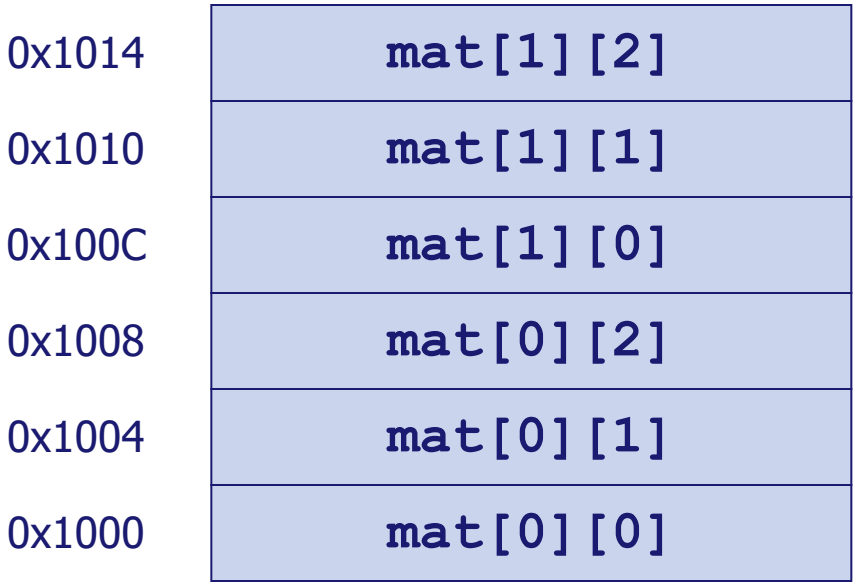
| | Column 0 | Column 1 | Column 2 | Column 3 |
|---|---|---|---|---|
| Row 0 | a[ 0 ][ 0 ] | a[ 0 ][ 1 ] | a[ 0 ][ 2 ] | a[ 0 ][ 3 ] |
| Row 1 | a[ 1 ][ 0 ] | a[ 1 ][ 1 ] | a[ 1 ][ 2 ] | a[ 1 ][ 3 ] |
| Row 2 | a[ 2 ][ 0 ] | a[ 2 ][ 1 ] | a[ 2 ][ 2 ] | a[ 2 ][ 3 ] |

# Memory Representation – 2D Arrays

```
int  mat[2][3];

mat[1][0] = 17;
```

**Valid Initialization**



# 2D Arrays

A[3][3]={1,2,3,4,5,6,7,8,9};
A[3][3]={ {1,2,3},{4,5,6},{7,8,9} };  // array of 1D array
a[0][1]  addr is  a[0][0] addr + Size of Data Type

| | Col 0 | Col 1 | Col 2 |
|---|---|---|---|
| Row 0 | 1 | 2 | 3 |
| Row 1 | 4 | 5 | 6 |
| Row 2 | 7 | 8 | 9 |

| | Col 0 | Col 1 | Col 2 |
|---|---|---|---|
| Row 0 | 1 4000 | 2 4004 | 3 4008 |
| Row 1 | 4 4012 | 5 4016 | 6 4020 |
| Row 2 | 7 4024 | 8 4028 | 9 4032 |

| [0][0] | [0][1] | [0][2] | [1][0] | [1][1] | [1][2] | [2][0] | [2][1] | [2][2] |
|---|---|---|---|---|---|---|---|---|
| **Base Address** | **+0** | **+4** | **+8** | **+12** | **+16** | **etc.** | | |

# Initializing 2D Arrays

```
#include<stdio.h>
int main()
 {
        int i, j;
        int a[3][3] = {
                        { 1, 4, 6 },
                        { 5, 2 ,7},
                         { 6, 5 ,8}
                };
 for (i = 0; i < 3; i++)
 {
        for (j = 0; j < 3; j++)
        {
                printf("%d ", a[i][j]);
        }
        printf("\n");
 }
        return 0;

 }
```

```
1 4 6
5 2 7
6 5 8
```

# 2D Arrays

A[3][3]={ {1,2,3},{4,5,6},{7,8,9} };  // array of 1D arrays

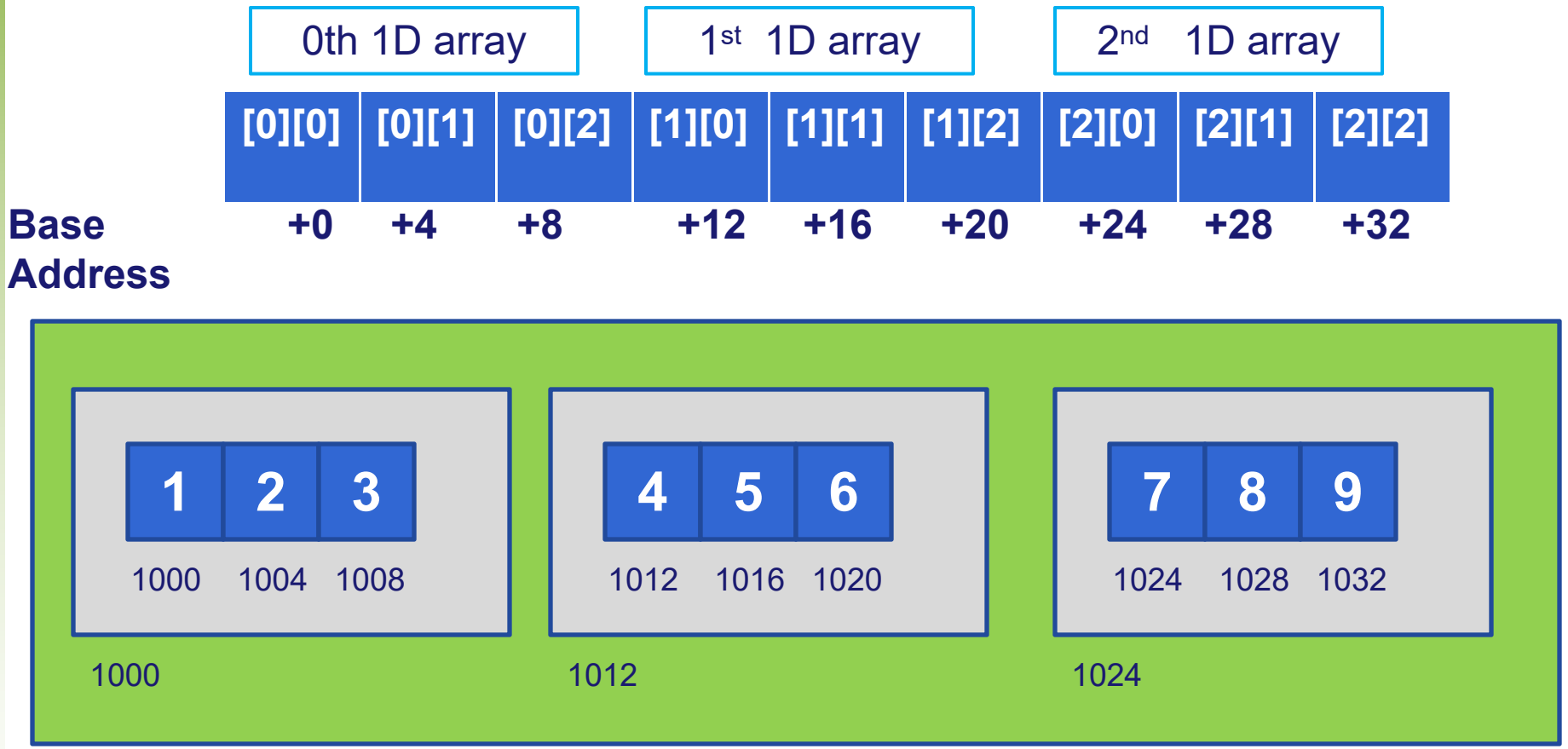
| | Col 0 | Col 1 | Col 2 |
|---|---|---|---|
| Row 0 | 1 | 2 | 3 |
| Row 1 | 4 | 5 | 6 |
| Row 2 | 7 | 8 | 9 |

| | Col 0 | Col 1 | Col 2 |
|---|---|---|---|
| Row 0 | 1 4000 | 2 4004 | 3 4008 |
| Row 1 | 4 4012 | 5 4016 | 6 4020 |
| Row 2 | 7 4024 | 8 4028 | 9 4032 |

| | 0th 1D array | | | 1st 1D array | | | 2nd 1D array | | |
|---|---|---|---|---|---|---|---|---|---|
| | [0][0] | [0][1] | [0][2] | [1][0] | [1][1] | [1][2] | [2][0] | [2][1] | [2][2] |
| **Base Address** | +0 | +4 | +8 | +12 | +16 | +20 | +24 | +28 | +32 |

# 2D Arrays

A[3][3]={ {1,2,3},{4,5,6},{7,8,9} };  // array of 1D arrays

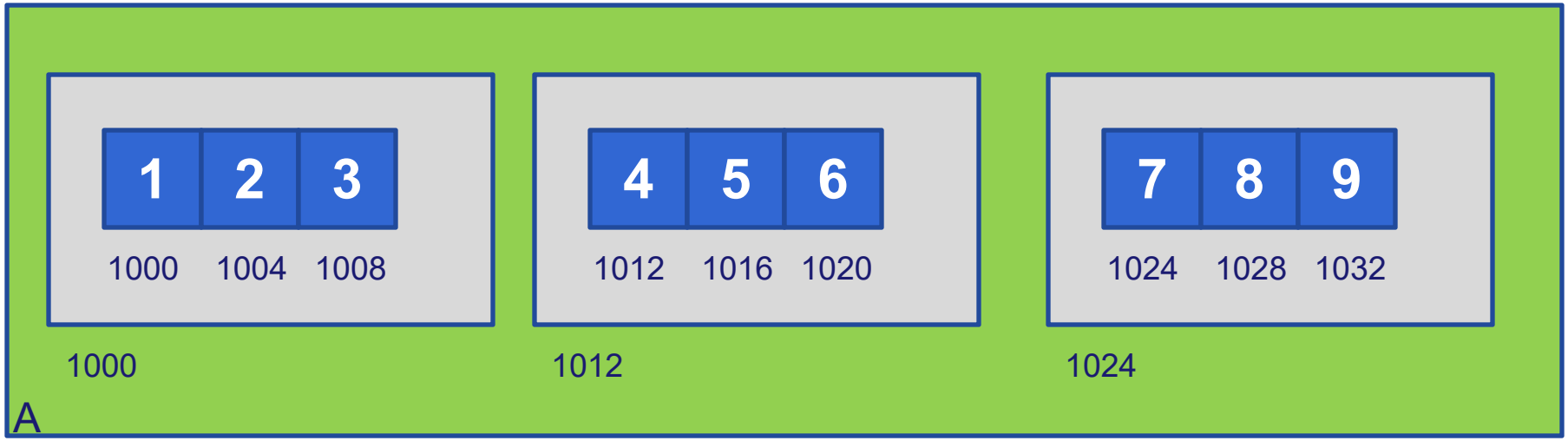



A points to 0th 1-D array.
(A + 1) points to 1st 1-D array.
(A + 2) points to 2nd 1-D array.

# 2D Arrays

A[3][3]={ {1,2,3},{4,5,6},{7,8,9} }; // array of 1D arrays

A points to 0th 1-D array. (A + 1) points to 1st 1-D array. (A + 2) points to 2nd 1-D array.




(A + 1) points to 1st 1-D array ….. A+1 is points 1st element is 1012

*(A+ 1) points to the address of the 0th element of the 1st 1-D array.

*(A+ 1) + 1 points to the address of the 1st element of the 1st 1-D array

*(A + i) + 2 points to the address of the 2nd element of the 1st 1-D array

*(*(A + 1) + 0) -- 1st 1-D array's 0th element … i.e A[1][0]

*(*(A + 1) + 1) -- 1st 1-D array's 1st element . … i.e A[1][1]

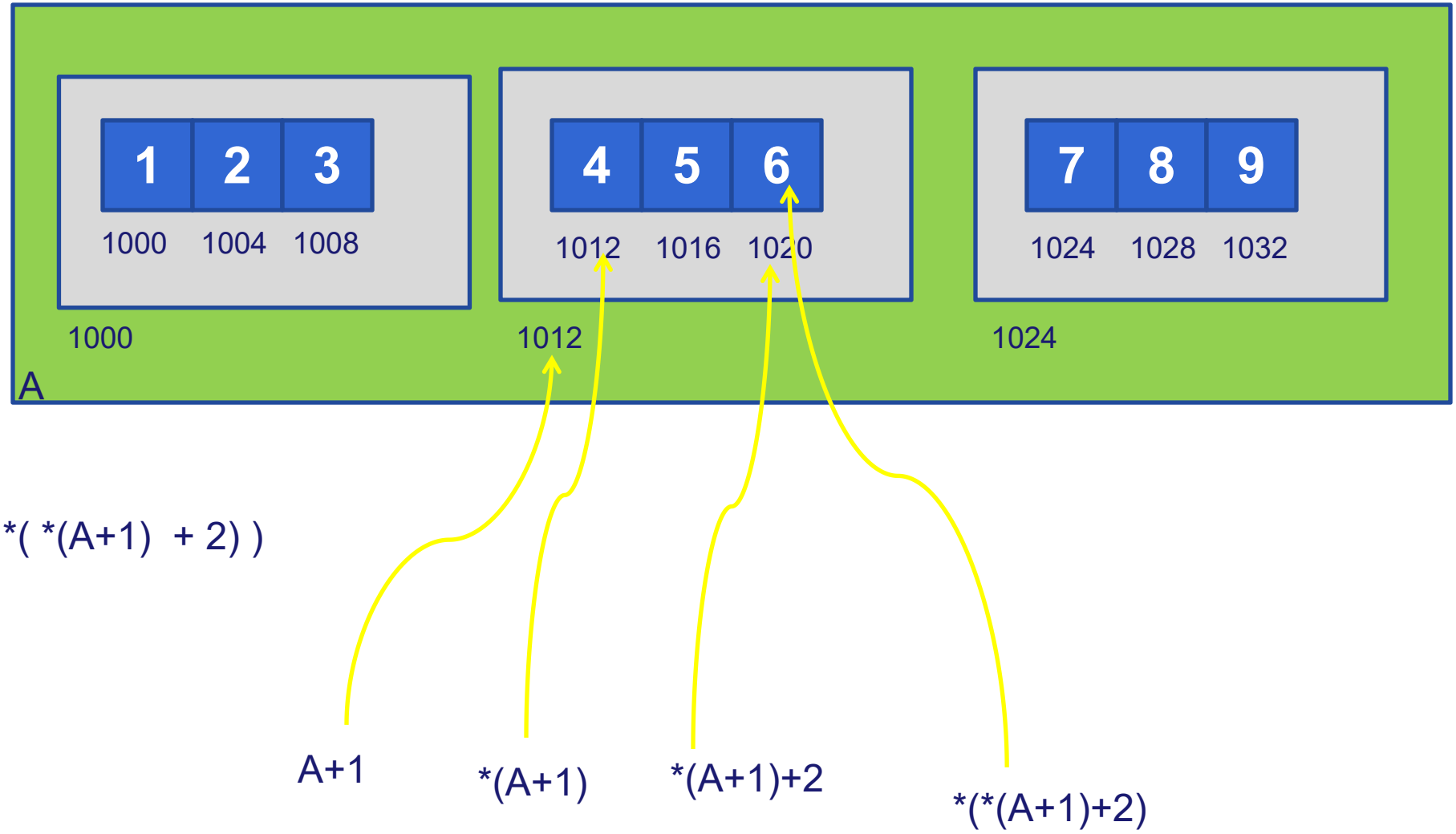
*(*(A + 1) + 2) -- 1st 1-D array's 2nd element . … i.e A[1][2]

***(*(A + i) + j) is same as A[i][j]**

# 2D Arrays

A[3][3]={ {1,2,3},{4,5,6},{7,8,9} };  // array of 1D arrays

A points to 0th 1-D array.  (A + 1) points to 1st 1-D array. (A + 2) points to 2nd 1-D array.



# Initializing 2D Arrays

```
#include<stdio.h>
int main ()
{
  int i, j;
  int a[3][3] = {      {1, 2, 3},      {4, 5, 6},      {7, 8, 9}    };
  for (i = 0; i < 3; i++)
    {
      for (j = 0; j < 3; j++)
        {
          printf ("%d and its address is %p ......", a[i][j], &a[i][j]);
          printf ("\n");
        }
      printf ("\n");
    }

  printf (" first 1 d array address  %p \n", a + 1);
  printf (" first 1 d array 0th element address  %p \n", *(a + 1));
  printf (" first 1 d array 2th element address  %p \n", *(a + 1) + 2);
  printf (" first 1 d array 2th element   %d \n", *(*(a + 1) + 2));

 return 0;
```

# Initializing 2D Arrays

```
#include<stdio.h>
int main ()
{
  int i, j;
  int a[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
  };

  printf ("  array base address  %u \n", a );

  printf (" first 1 d array address  %u \n", a + 1);
  printf (" first 1 d array 0th element address  %u \n", *(a + 1));
  printf (" first 1 d array 2th element address  %u \n", *(a + 1) + 2);
  printf (" first 1 d array 2th element   %d \n", *(*(a + 1) + 2));

  printf (" first 2 d array address  %u \n", a + 2);
  printf (" first 2 d array 0th element address  %u \n", *(a + 2));
  printf (" first 2 d array 1th element address  %u \n", *(a + 2) + 1);
  printf (" first 2 d array 1th element   %d \n", *(*(a + 2) + 1));
  return 0;
}
```

# Array of Pointers

- Just like array of integers or characters, there can be array of pointers too. An array of pointers can be declared as :

  <type> *<name>[<number-of-elements];

- For example :

  int *ptr[3];

# What is the output of following program:

```
#include <stdio.h>
void main ()
{
  int num[] = { 10, 20, 30 };
  int i, *ptr[3];

  for (i = 0; i < 3; i++)
    {
      ptr[i] = &num[i];          /* assign the address of integer. */
    }
  for (i = 0; i < 3; i++)
    {
      printf ("num[%d] = %d\n", i, *ptr[i]);
    }
}
```

# Array of Pointers

## What is a pointer array?

- An array can store pointers i.e addresses
- We can use array of pointers in a similar way to a multi dimensional array
- **Example1:** Store and print the names of three students

```
#include<stdio.h>
int main()
{
                int i;
                char *students[]={"Ram", "John", "Abdul"};
                for (i=0; i<3; i++)
                {
                        printf("%s\n", students[i]);
                }
return 0;
}
```

# Array of Pointers

**Example 2:** Write a function which returns a pointer to a character string containing the name of the nth month.

```
char * month_name (int n)
{
  static char *name[] = {
    "Illegal month",
    "January", "February", "March",
    "April", "May", "June",
    "July", "August", " September ",
        " October ", " November ", " December "
                          };
return (n < 1 || n > 12) ? name[0] : name[n];
}
void main()
{
int i;
for(i=0;i<=12;i++)
printf(" %s\n ", month_name(i));

}
```

# Array of Pointers…

**Memory Representation**