COMPUTER PROGRAMMING

Pointers and Strings

Topics to be covered

Reading, writing and manipulating Strings, Understanding computer memory, accessing via pointers, pointers to arrays, dynamic allocation, drawback of pointers.

String definition

- String is a 1-d array of characters terminated by a null (‘\0’)
- char C[] = {'A','B','C','\0'};
- char S[] = "ABC";
- Both C and S have length = 3

String input including blanks

```
main(){
char name[25] ;
printf ( "Enter name: " ) ;
scanf ( "%[^\n]s", name ) ;
printf("hello %s",name);
}
```

Another way

```
main(){
char C[20];
gets(C);   // accepts blank spaces
puts(C);
}
```

NULL and strlen()

```
main(){
int i=0;
char C[10];
gets(C);
while(C[i]!=NULL) printf("%c",C[i++]);
//OR while(i<strlen(C)) printf("%c",C[i++]);
}
```

Few Inbuilt functions

```
main( ) {
char A[] = "wxYZ", B[]="78",C[20];
printf("length of %s = %d",A,strlen(A));
if(strcmp(A,B)==0)  printf("\nA,B same srings");
printf("\nC = %s",strcpy(C,A));
printf("\nUppercase of A = %s",strupr(A));
printf("\nConcat A+C = %s",strcat(C,A));
printf("\n Reverse A = %s",strrev(A));
}
```

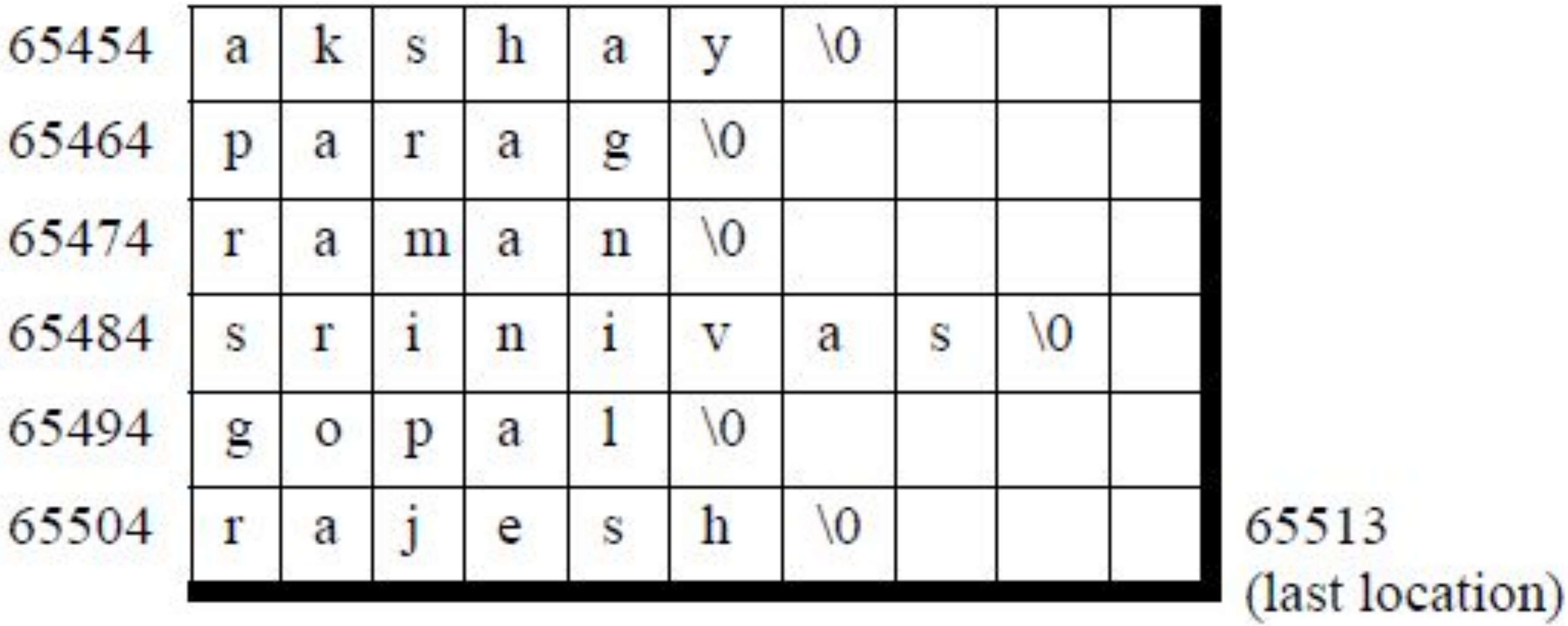
Reverse of string without strrev()

```
int main() {
   char str[10], temp;
   int i=0, j = 0;
    printf("\nEnter the string :");
   gets(str);
   while(str[i]!=NULL)//String Length
   {    i++;  j++;    }
  j=j-1;i=0;

  while (i < j) {    //String Reverse
      temp = str[i];
      str[i] = str[j];
      str[j] = temp;
      i++;      j--;
   }
    printf("\nReverse string is :%s", str);
   return (0);
}
```

2-d character array

```
char Names[6][10] = {"akshay","parag","raman"};

puts(Names[0]);  //akshay
```

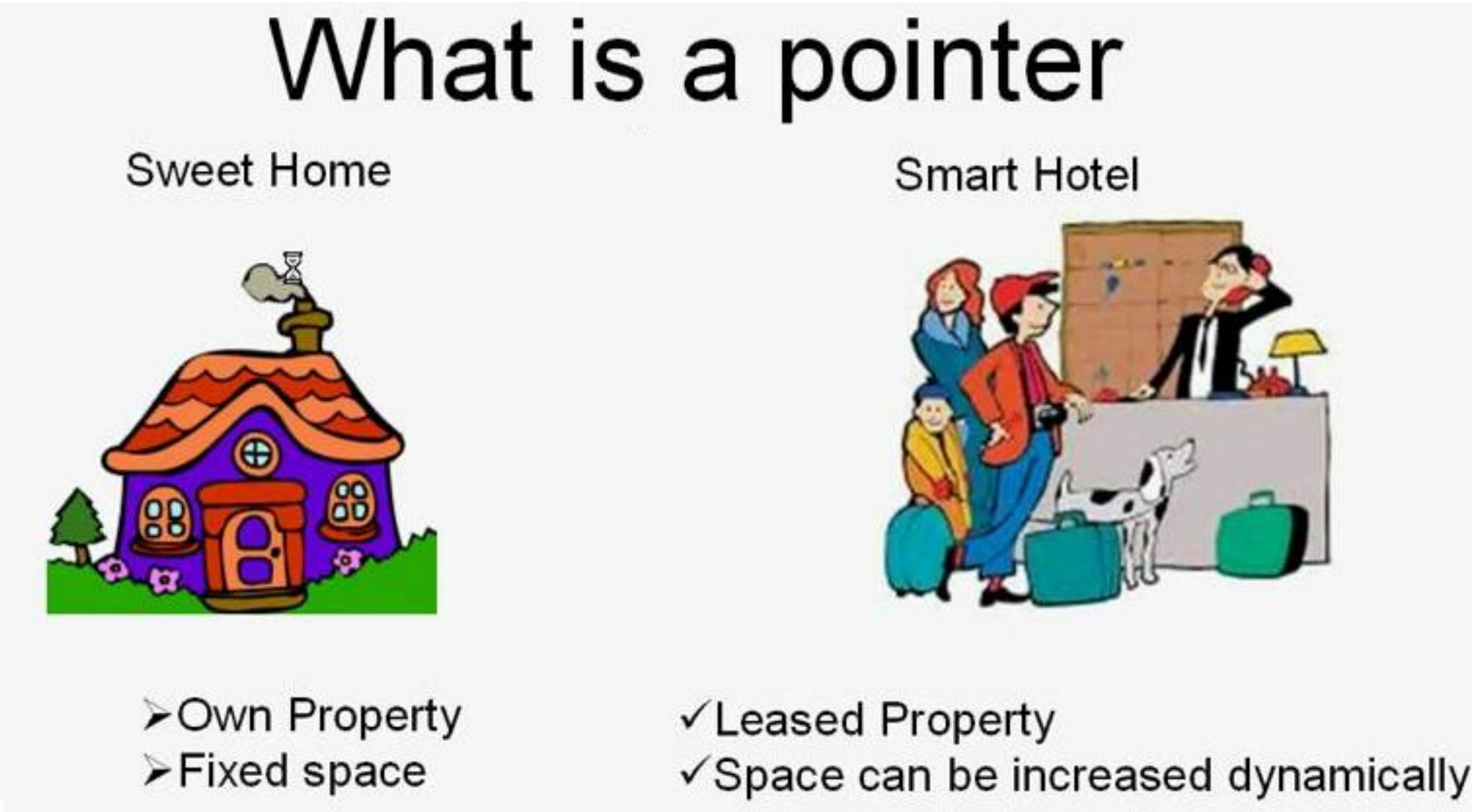
char Names[6][10] storage



65454	a	k	s	h	a	y	\0			
65464	p	a	r	a	g	\0				
65474	r	a	m	a	n	\0				
65484	s	r	i	n	i	v	a	s	\0	
65494	g	o	p	a	l	\0				
65504	r	a	j	e	s	h	\0			

65513
(last location)

- Reading, writing and manipulating Strings, Understanding computer memory, accessing via pointers, pointers to arrays, dynamic
allocation, drawback of pointers.



What is a pointer

Sweet Home

➢Own Property
➢Fixed space

Smart Hotel

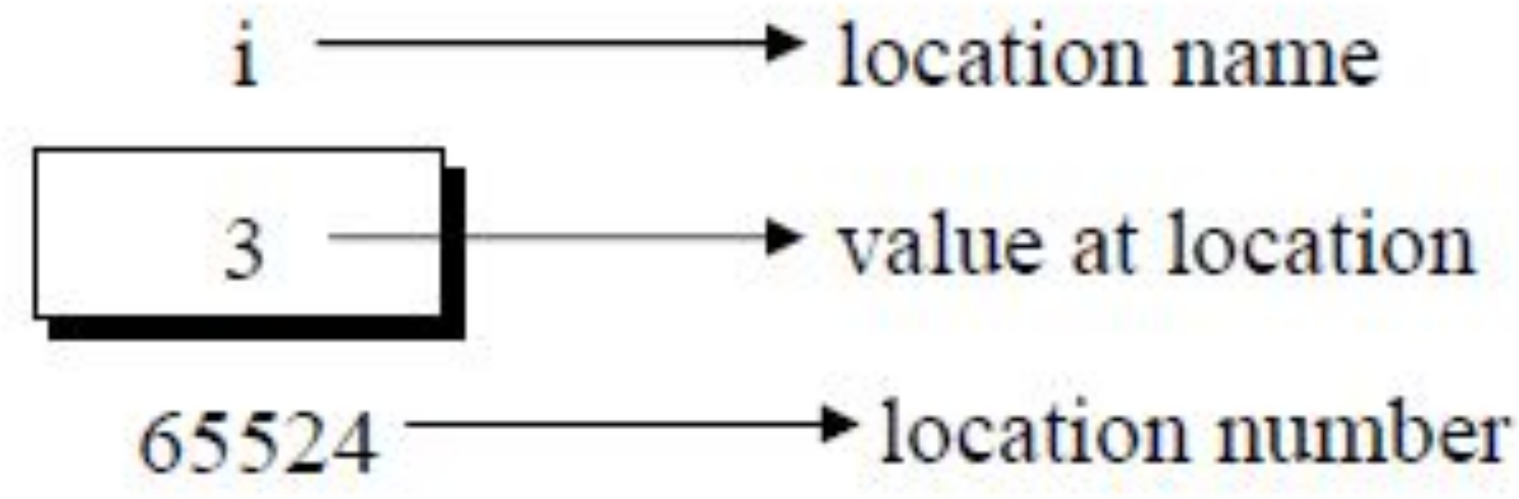
✓Leased Property
✓Space can be increased dynamically

Variables vs. pointers

Advantages of pointers

- Dynamic memory allocation
- System level programming
- To modify a variable in another function
- Illusion of multiple returns
- Passing and returning arrays

Variable, value and location



i → location name

3 → value at location

65524 → location number

& - address of operator

```
main( ){
int i = 3 ;
printf ( "\nAddress of i = %u", &i ) ;
printf ( "\nValue of i = %d", i ) ;
}

//%u means unsigned integer, what is %p,%i
```

* is value at address

```
main( ) {
int i = 3 ;
printf ( "\nAddress of i = %u", &i ) ;
printf ( "\nValue of i = %d", i ) ;
printf ( "\nValue of i = %d", *( &i ) ) ;
}
// output: some address, 3 and 3
```

int *j ;

- This declaration tells the compiler that j will be used to store the *address* of an integer value.

Passing Pointer to a Function

- In this example, we are passing a pointer to a function. When we pass a pointer as an argument instead of a variable then the address of the variable is passed instead of the value.
- So any change made by the function using the pointer is permanently made at the address of passed variable.

- This technique is known as call by reference in C**.**

```
#include <stdio.h>
void salaryhike(int  *var, int b)
{
   *var = *var+b;
}
```

```
int main()
{
   int salary=0, bonus=0;
   printf("Enter the employee current salary:");
   scanf("%d", &salary);
   printf("Enter bonus:");
   scanf("%d", &bonus);
   salaryhike(&salary, bonus);
   printf("Final salary: %d", salary);
   return 0;
}
```

Problem 1

```
main( ) {
int i = 3 ,*j ;
j = &i ;
printf ( "\nAddr of i = %u", &i ) ;
printf ( "\nAddr of i = %u", j ) ;
printf ( "\nAddr of j = %u", &j ) ;
printf ( "\n j = %u", j ) ;   printf ( "\n i = %d", i ) ;
printf ( "\n i = %d", *( &i ) ) ;  printf ( "\n i = %d", *j ) ;
}
```

Answer

```
Addr of i = 65524
Addr of i = 65524
Addr of j = 65522
j = 65524
i = 3
i = 3
Value of i = 3
```

Pointers for other data types

- int *alpha ;
- char *ch ;
- float *s ;

Pointers are variables that contain addresses, and since addresses are whole numbers, pointers would always contain whole numbers.

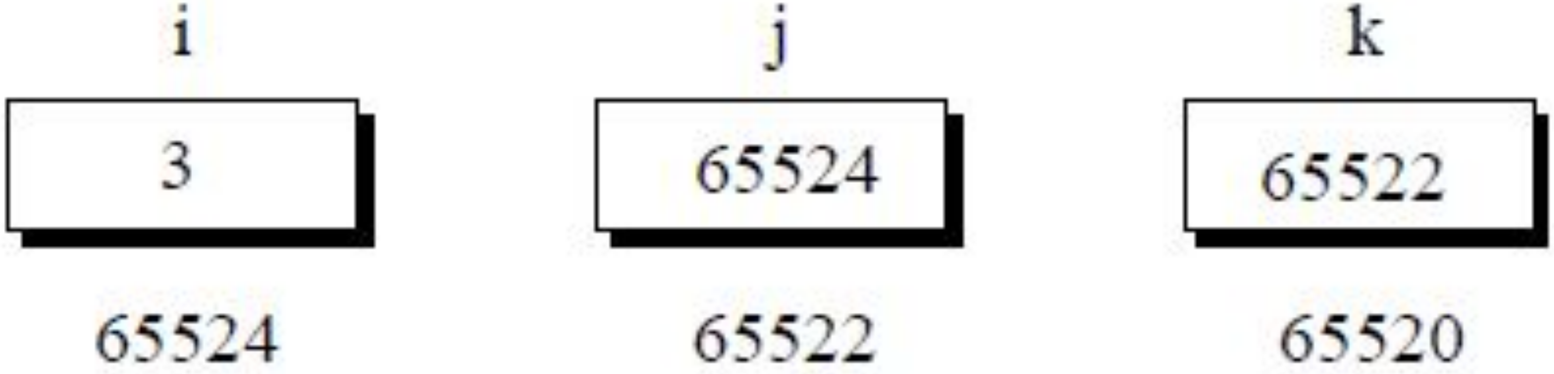
char *ch ;

- *ch will contain a character
- ch is an unsigned int containing the address

Problem 2

```
main(){
inti=3,*j,**k;
j=&i; k=&j;
printf("\naddr of i=%u",&i);
printf("\naddr of i=%u",j);
printf("\naddr of i=%u",*k);
printf("\naddr of j=%u",&j);
printf("\naddr of j=%u",k);
printf("\naddr of k=%u",&k);
printf("\nj=%u",j); printf("\nk=%u",k);
printf("\ni=%d",i); printf("\ni=%d",*(&i));
printf("\ni=%d",*j); printf("\ni=%d",**k);
}
```

Situation



Answer

```
addr of i=6487628
addr of i=6487628
addr of i=6487628
addr of j=6487616
addr of j=6487616
addr of k=6487608
j=6487628
k=6487616
i=3
i=3
i=3
i=3
```

Problem 3

```
main() {
int Var = 10;
int *ptr = &Var;
printf("Var = %d\n", *ptr);
printf("Var Addr = %u\n", ptr);
*ptr = 20;
printf("Var = %d\n", Var);
}
```

Answer

```
Var = 10
Var Addr = 6487620
Var = 20
```

Problem 4

```
main() {
int v[3] = {10, 100, 200},i;
int *ptr;
ptr = v;
for(i = 0; i < 3; i++)  {
printf("*ptr = %d\n", *ptr);
printf("ptr = %u\n", ptr);
ptr++;
    }
}
```

Answer

```
*ptr = 10
ptr = 6487600
*ptr = 100
ptr = 6487604
*ptr = 200
ptr = 6487608
```

Problem 5

```
main( ) {
int i=3,*x; float j=1.5,*y; char k='c',*z ;
printf("\n i=%d, j=%f, k=%c",i,j,k) ;
x = &i ; y = &j ; z = &k ;
printf("\n Addr of x,y,z = %u, %u, %u",x,y,z);
x++; y++; z++;
printf("\n New Addr of x,y,z = %u, %u,
    %u",x,y,z);
}
```

Answer

- i=3, j=1.500000, k=c
- Addr of x,y,z = 6487604, 6487600, 6487599
- New Addr of x,y,z = 6487608, 6487604, 6487600

Problem 6

```
main( ) {
int arr[] = { 10, 20, 30, 45, 67, 56, 74 };
int *i, *j;
i = &arr[1]; j = &arr[5];
printf("%d %d", j-i, *j-*i);
}
```

Answer

- 4 36

Problem 7

```
main(){
int arr[]={10,20,36,72,45,36};
int*j,*k;
j=&arr[4];
k=(arr+4);
if(j==k)
printf("Same locations");
else printf("Different locations");
}
```

Answer

- Same locations

Problem 8

```
main(){
int a = 10, b = 20 ;
swapv ( a, b ) ;
printf ( "\na = %d b = %d", a, b ) ;
}
swapv ( int x, int y ) {
int t  = x ;
x = y ; y = t ;
printf ( "\nx = %d y = %d", x, y ) ;
}
```

Answer

x = 20 y = 10

a = 10 b = 20

Problem 9: Pointer swap

```
main( ) {
int a = 10, b = 20 ;
swapr ( &a, &b ) ;
printf ( "\na = %d, b = %d", a, b ) ;
}
swapr( int *x, int *y ) {
int t  = *x ; *x = *y ; *y = t ;
}
```

Problem 10: One * for 1-d

```
main( ) {
int a[3]={1,2,3},*b,i;
b=a;
for(i=0;i<3;i++) {
printf("\n %d %d",a[i],b[i]);
}}
```

Output

- 1 1
- 2 2
- 3 3

Problem 11: Multiple returns

```
AreaPeri(int r, float *A, float *P){
*A = 3.14*r*r;  *P = 2*3.14*r;
}
main( ){
int r;  float A,P;
printf("\nEnter radius: " );   scanf("%d",&r);
AreaPeri(r,&A,&P);  printf("Area = %f, Peri = %f",A,P);
}
```

Same things

- name[i]
- *(name + i)
- *(i + name)
- i[name]

Problem 12: Array vs *

```
main( ){
char A[] = "Hello", B[10] ;
char *C = "Good Morning", *D;
//B = A ; /* error */
D = C ; /* Okay */
printf("%s",D);
} // cannot assign array to array, but can assign
   pointer to pointer
```

Problem 13: * are dynamic

```
main( ) {
char A[] = "Hello" ;
 char *B = "Hello" ;
//A = "Bye" ; /* error */
B = "Bye" ; /* works */
printf("%s",B);
}
```

Problem 14: Returning an array

```
int *incr(int X[3],int k){
int i;
for(i=0;i<k;i++) ++X[i];
return X;
}
main(){
int A[] = {1,2,3},n=3,i,*B;
B = incr(A,n);
for(i=0;i<n;i++) printf(" %d",B[i]);
}
```

Output

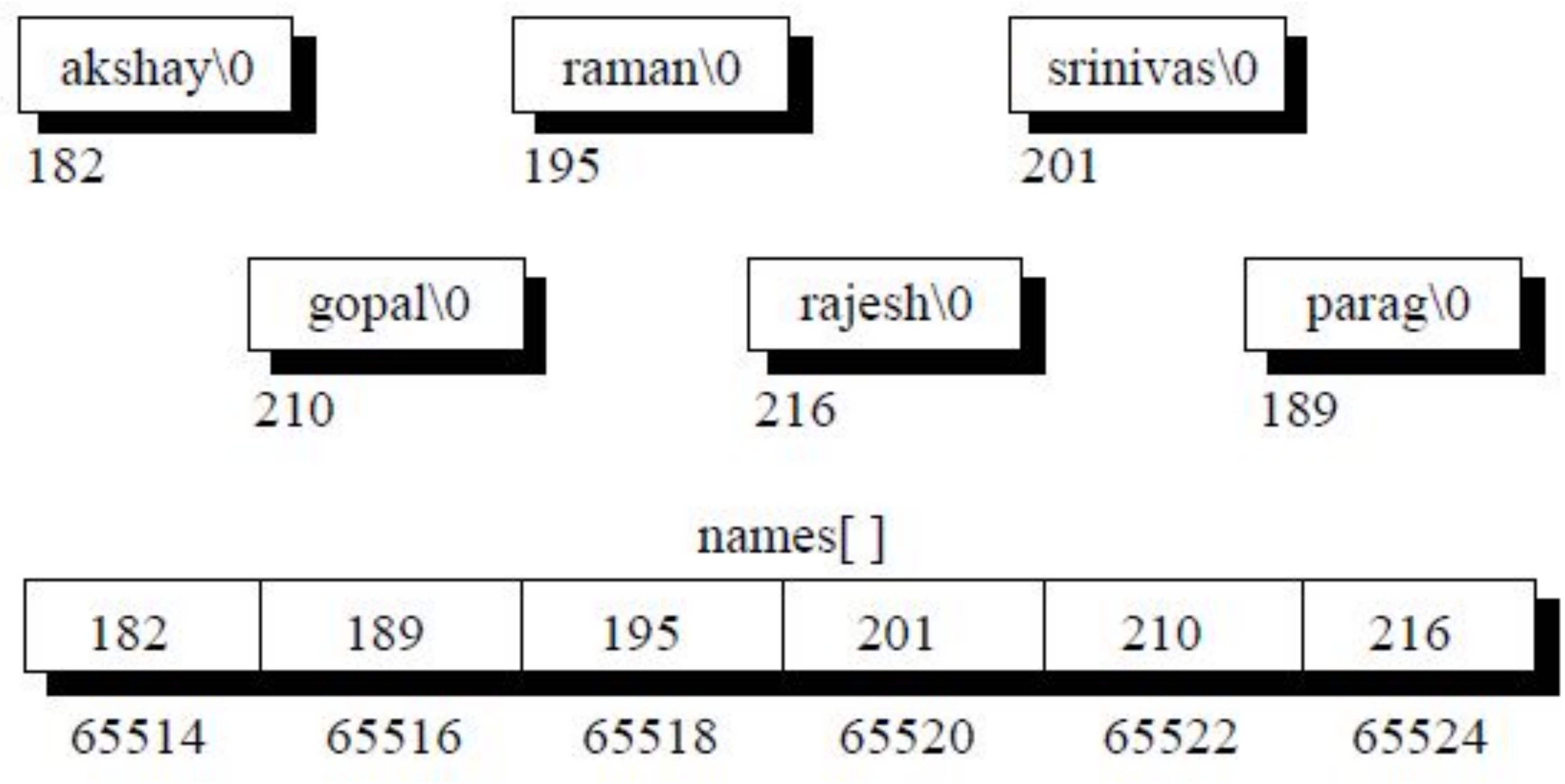
- 2 3 4

Array of pointers

- char*names[]={"akshay","parag","raman","srinivas", "gopal","rajesh"};

- It contains base address of various names

Array of pointers



akshay\0
182

raman\0
195

srinivas\0
201

gopal\0
210

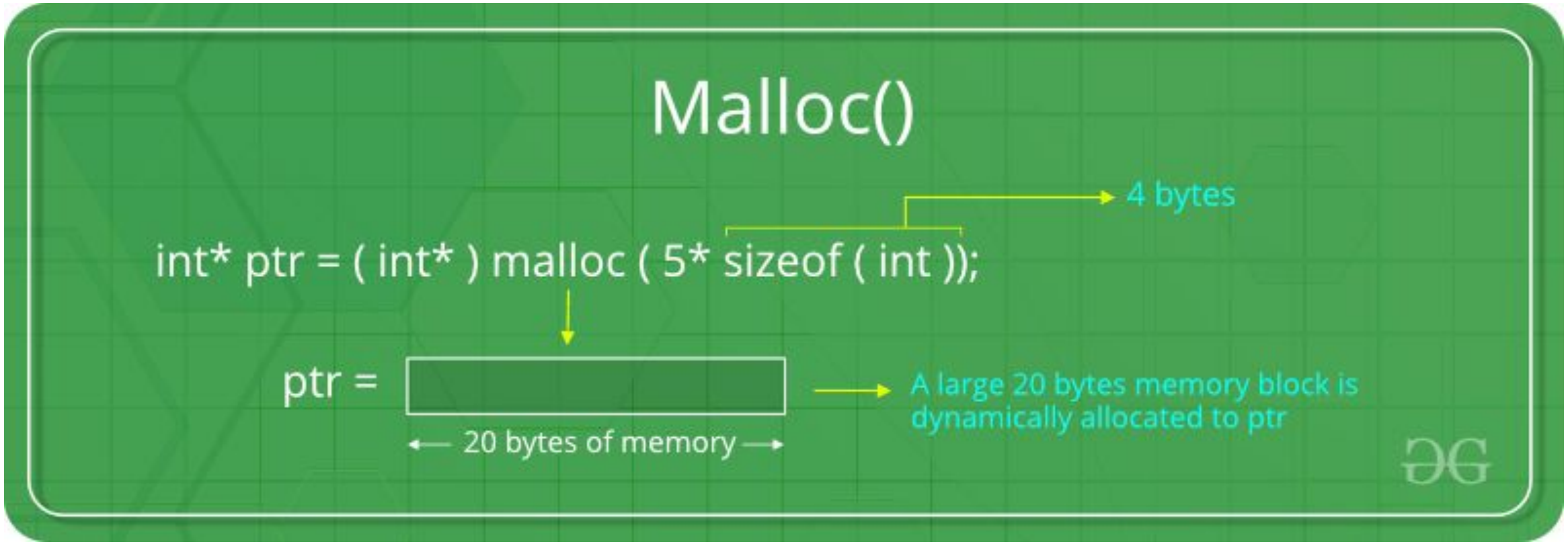
rajesh\0
216

parag\0
189

names[ ]

182	189	195	201	210	216
65514	65516	65518	65520	65522	65524

Malloc() in C



Malloc()

4 bytes

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

ptr =

20 bytes of memory

A large 20 bytes memory block is dynamically allocated to ptr

Problem 15: Dynamic allocation

```
main(){
int *ptr, n,i;
printf("Size of array? "); scanf("%d",&n);
ptr = (int*)malloc(n*sizeof(int));
for(i=0;i<n;i++) ptr[i]=i;
for(i=0;i<n;i++) printf(" %d",ptr[i]);
free (ptr);
}
```

Output

```
Size of array? 10
 0 1 2 3 4 5 6 7 8 9
```

Drawback of pointers

- Complicated to use and debug
- Uninitialized pointers might cause segmentation fault
- Dynamically allocated block needs to be freed explicitly. Otherwise, it would lead to **memory leak** (occupied but unused memory)
- If pointers are updated with incorrect values, it might lead to memory corruption