Programming & Data Structure Laboratory

Arrays, pointers

August 6, 2015

€ Pointers and Multidimensional Array

Counting function calls in Fibonacci

```
#include<stdio.h>
int total_call=0; /* Declare global variable*/
int Fib(int n){
total_call++;  if(n==0)  return 0;   if(n==1)  return 1;
return(Fib(n-1)+Fib(n-2));
}
int main(void){
int n, fib_val;
printf("\n n = "); scanf("%d",&n);
fib_val=Fib(n);
printf("\n value = %d and no. of recursive calls=%d\n",
    fib_val,total_call);}
```

Pointers

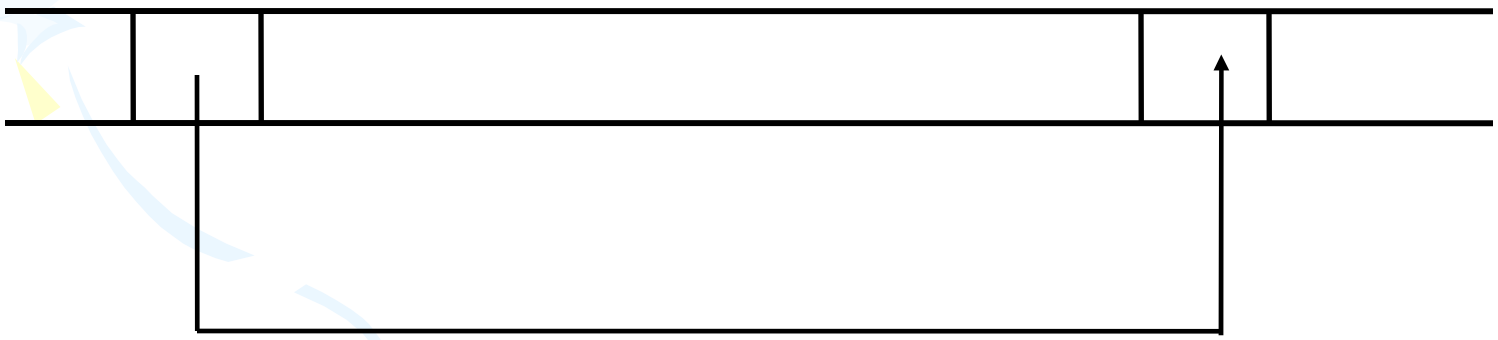
int *p

int x

p = &x

int *p

int x



What is *p? If x = 5, what is *p = ?

An example - swap

```
#include<stdio.h>
void swap(int* x, int* y)
{  int temp;
   temp=*x;*x=*y;*y=temp;
   return;
}

int main(void) {
int a=5,b=4;
swap(&a,&b);
printf("\na=%d, b=%d \n",a,b);
return 0;
}
```

Pointers and sizeof operator

```
#include<stdio.h>


int main(void)
{
  printf("\n The size of pointer to char = %ld \n", sizeof(char *),);
  printf("\n The size of pointer to uchar = %ld \n",
    sizeof(unsigned char *));
  printf("\n The size of pointer to int = %ld \n", sizeof(int *));
  printf("\n The size of pointer to long double = %ld \n",
    sizeof(long double *));
  return 0;
}
```

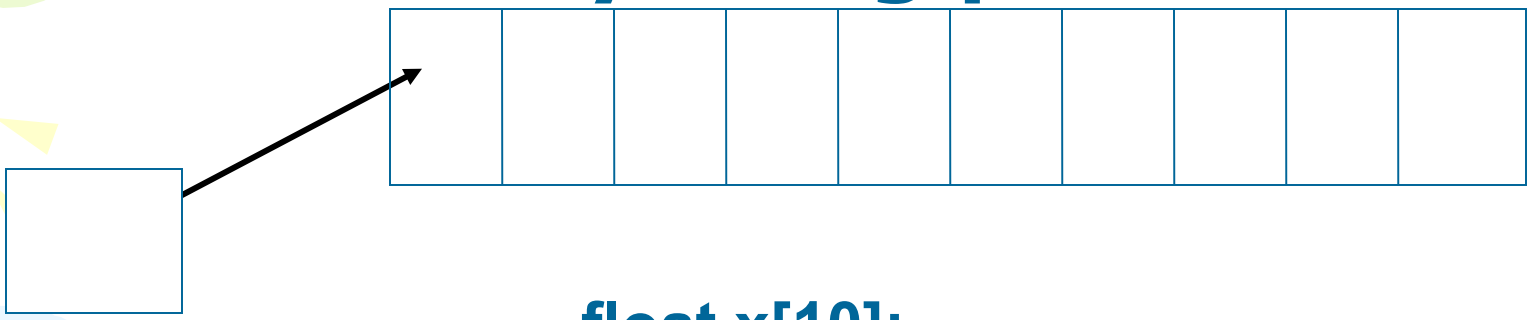
- Write a program to count the number of calls in the recursive version of Fibonacci number computation without using global variables. Use pointers and pass it to functions.

Fib. recursive without global variable

```
#include<stdio.h>
int Fib(int n, int* total_call)  {
(*total_call)++;
if(n==0)  return 0;   if(n==1)  return 1;
return (Fib(n-1,total_call)+Fib(n-2,total_call));
}

int main(void)  {
int n, fib_val, total_call=0;
printf("\n n = "); scanf("%d",&n);
fib_val=Fib(n,&total_call);
printf("\n value = %d and no. of recursive calls=%d\n",
    fib_val,total_call);}
```

1D array using pointers



float *p;

float x[10];

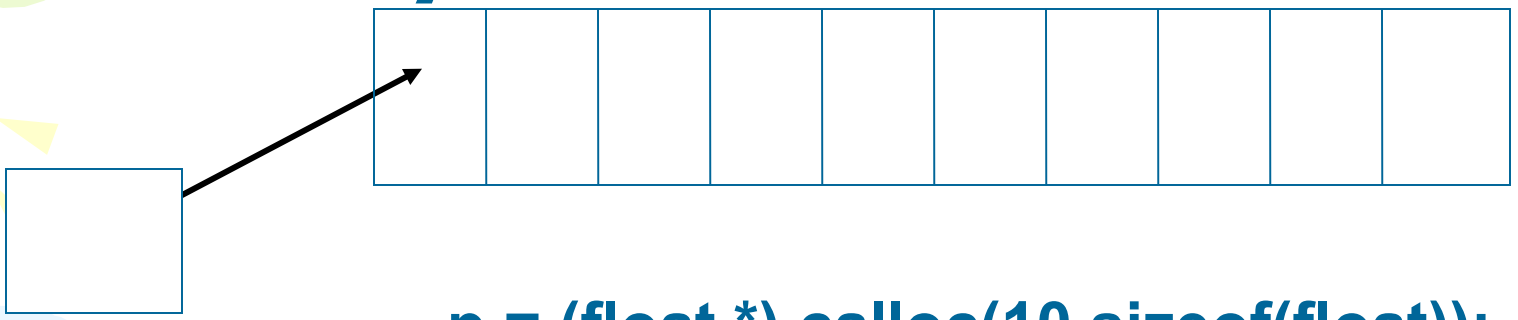
p = &x[0];

Show the pointers correctly for the following statements:

p = &x[6];

p = p+2;

Dynamic allocation



float *p;

p = (float *) calloc(10,sizeof(float));

Show the pointers correctly for the following statements:

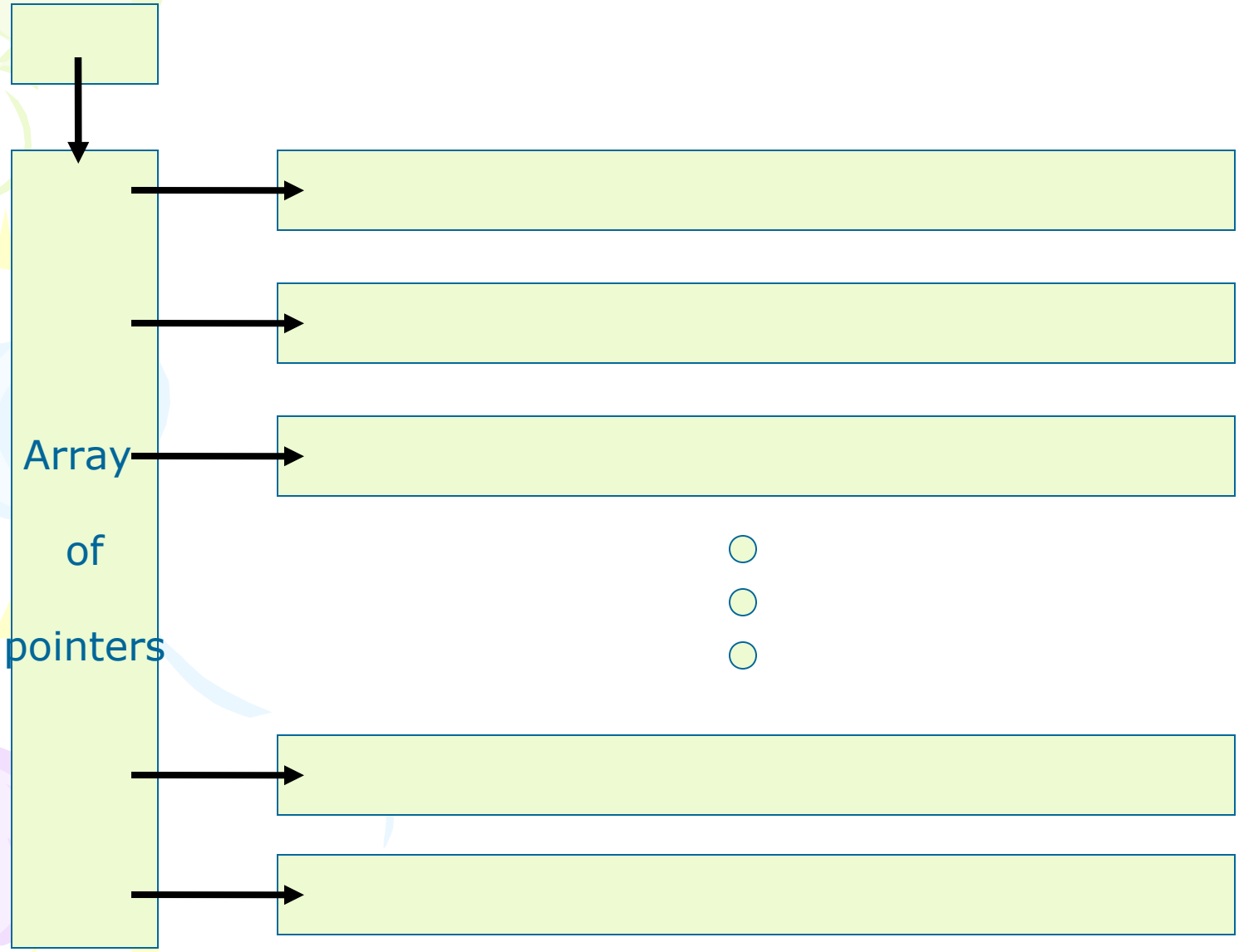
p = &p[6];

p = p+2;

```
#include<stdio.h>
#include<stdlib.h>
int* allocate1D(int row){
int *S;
S=(int *)calloc(row,sizeof(int));
if(S==NULL) { printf("\n No space \n"); exit(0);  }
return S;
}

int main(void){
int *S1,row;
printf("\n How many rows? "); scanf("%d", &row);
S1=allocate1D(row);
return 0;
}
```

Dynamic allocation of 2D array



- Write a small piece of code to allocate a two dimensional matrix using pointer to pointer.

```
#include<stdio.h>
#include<stdlib.h>
int** allocate2D(int row,int col){  int **S,k;
S=(int **)calloc(row,sizeof(int *));
if(S==NULL)  {  printf("\n No space \n"); exit(0);  }
for(k=0; k<row; k++)  {
S[k]=(int *)calloc(col,sizeof(int));
if(S[k]==NULL)   { printf("\n No space \n"); exit(0);  }
}
return S;
}
int main(void){  int **S2,row,col;
printf("\n How many rows? "); scanf("%d",&row);
printf("\n How many columns? "); scanf("%d",&col);
S2=allocate2D(row,col);
return 0;
}
```

- Write a small piece of code for multiplying two two-dimensional matrices using pointer to pointer and passing them as arguments to a function.