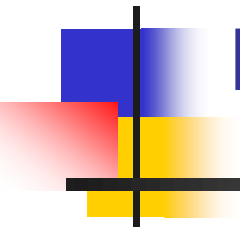
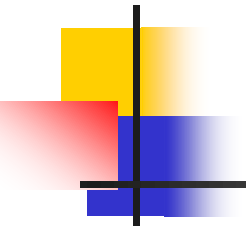


Program Patterns:
Basic C Review

J.W. Choi

2024



Roadmap

- Arrays and functions
- Pointer variables
- Function call by reference

Arrays and Functions

Passing an Array to a Function

- Pass the array name (i.e., starting address of the first element of the array) as argument.
  - saves memory and copying time
- The original array can be changed by the function.
- Must be very very careful with array overflow and underflow.

Syntax

- Function Prototype
  - return_data_type function_name (array_data-type [MAXELEMENTS]);
- Function Definition (Parameter)
  - data_type array_name [ ]
  - (Array size is not necessary, since it is defined in the function prototype.)
- Function Call (Argument)
  - array_name
  - ([ ] is not necessary, since only the starting address is being passed to the function.)
  - example: puts (message)

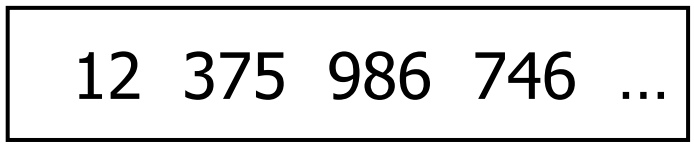
Example 1

```
char message[81] = “Program”;
char names[10][20] = {“Hong Gil Dong”};

puts (message);
puts (names[k]);
```

Example 2

```
#define MAXNUM 1000
void findMax (int [MAXNUM]);

void main ()
{
    int numList[MAXNUM];
    ....
    findMax(numList);
    ....
}
void findMax (int  local_array[])
{
    int i, max = local_array[0];

    for (i=1; i<MAXNUM; i=i+1)
      if (max < local_array[i])
          max = local_array[i];
    printf ("%d", max);
}
```

Logical View




```
#define MAXNUM 1000
void findMax (int [MAXNUM]);

void main ()
{
    int numList[MAXNUM];
    ....
    findMax(numList);
    ....
}
void findMax (int  local_array[])
{
    int i, max = local_array[0];

    for (i=1; i<MAXNUM; i=i+1)
      if (max < local_array[i])
          max = local_array[i];
    printf ("%d", max);
}
```

Example 3

```
#define MAXNUM 1000
int findMax (int [MAXNUM]);

void main ()
{
   int numList[MAXNUM];
   ....
   newmax = findMax(numList);
   ....
}
int findMax (int  local_array[])
{
   int i, max = local_array[0];

   for (i=1; i<MAXNUM; i=i+1)
     if (max < local_array[i])
         max = local_array[i];
   return max;
}
```

Example 4 (Function Definition Only)

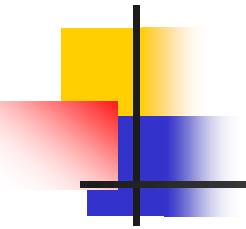
```
void strcopy(char string1[], char string2[])
{
    int i=0;
    while (string2[i]) != '₩0')
    {
        string1[i] = string2[i];
        i=i+1;
     }
     string[i] = '₩0';

    /*  print string1  */
}
```

Exercise

- Write a C program that stores an array of miles, and calls a function named mile2km.
- The mile2km function takes an array of miles, converts it to kilometers, and prints it.
  - 1 mile = 1.6093 kilometers

Solution

```
#define MAXNUM 5
void main ()
{
    void mille2km (int [MAXNUM]);
    float miles[MAXNUM];
    /*  store into the miles array  */
    miles2km (miles);
}

void mile2km (float  local_array[])
{
    int i;

    for (i=1; i<MAXNUM; i=i+1)
        printf ("%f", local_array[i] * 1.6093);
}
```

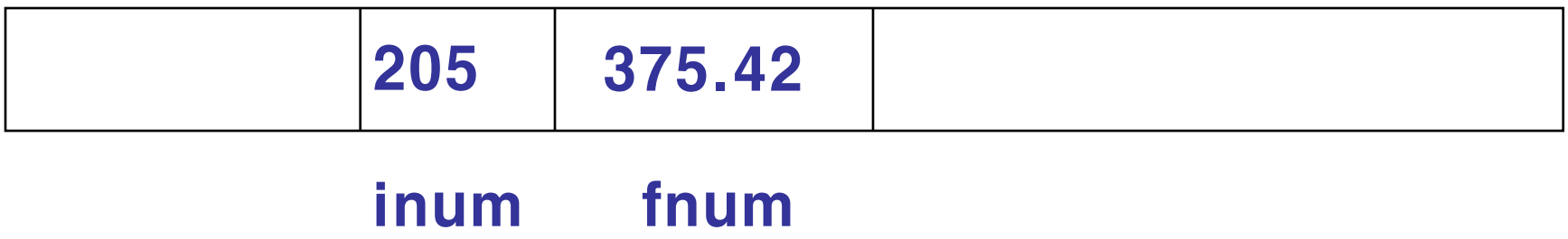


Exercise

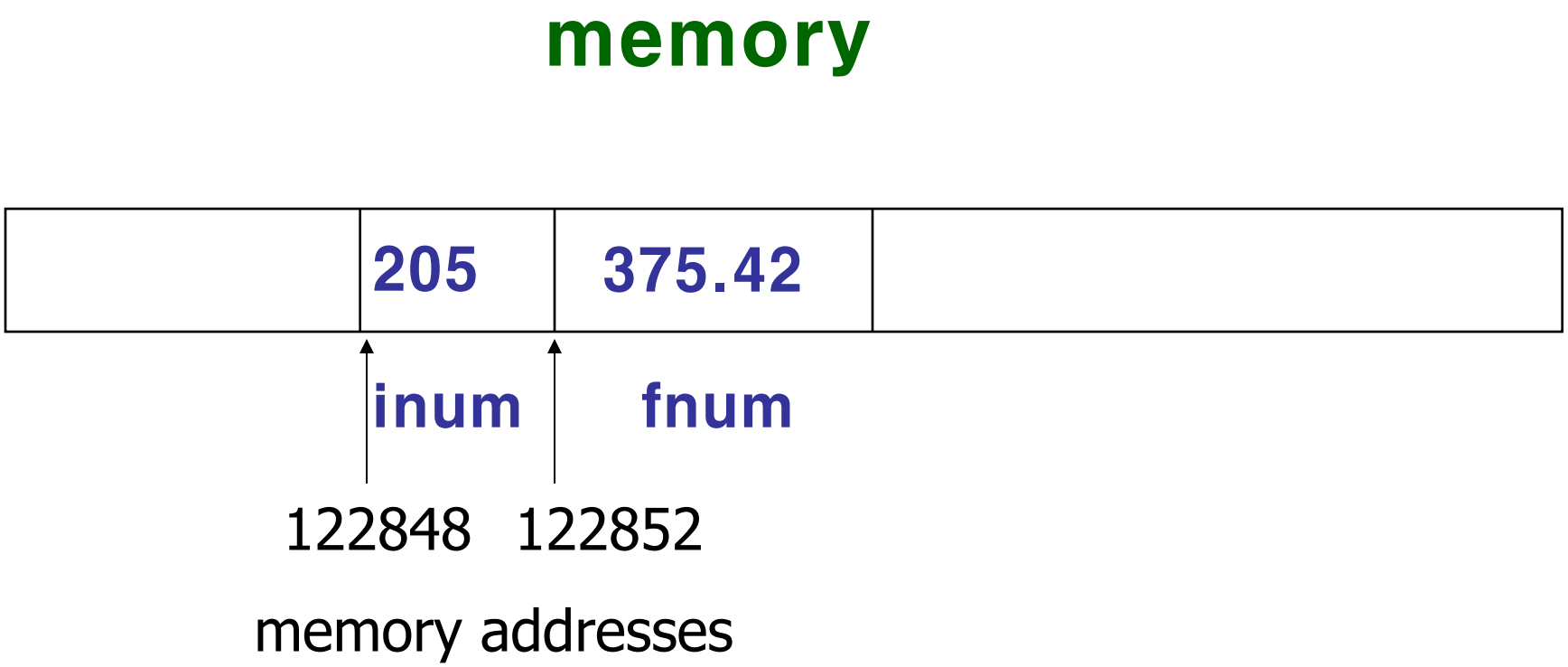
- Write a C program that stores a string, and calls a function named word_count .
- The word_count function takes a string input, and returns the number of words in the string.
- (no need to write the function body of word_count.)

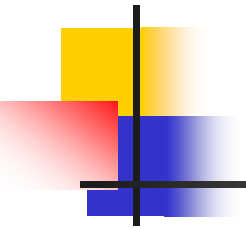
Pointer Variables

Storing Data in Memory



Storing Data and Memory Addresses

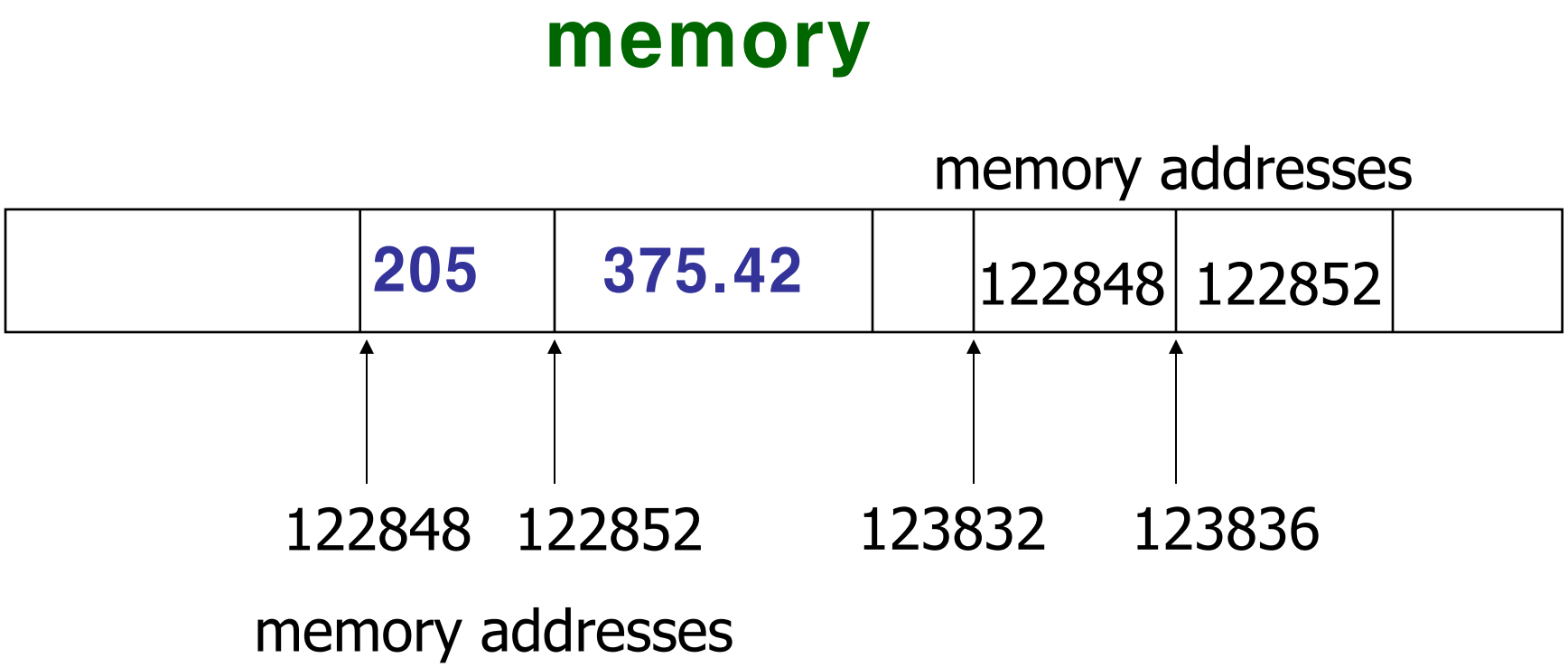




Pointer Variable

- Stores the memory address of a variable

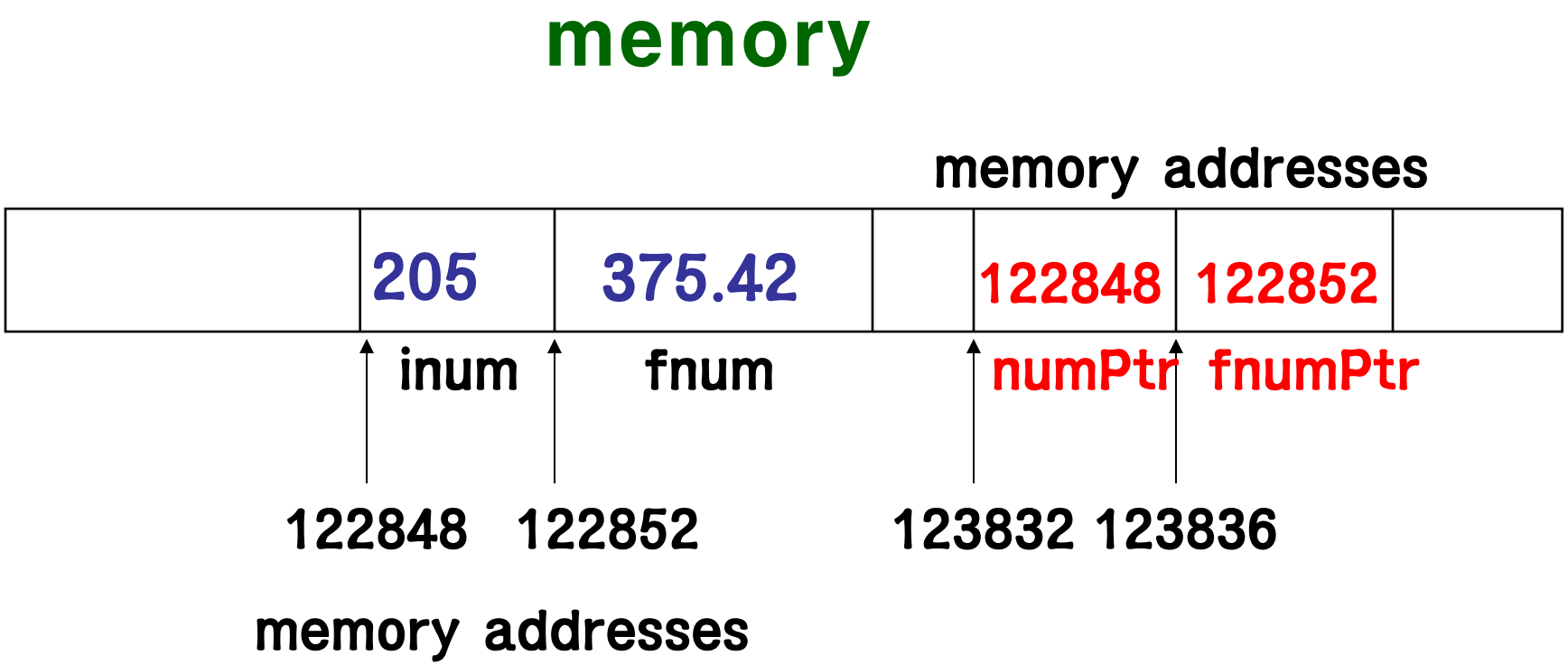
Storing Data and Memory Addresses

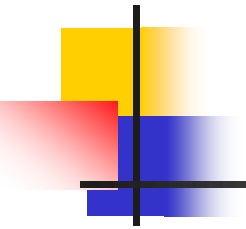


Pointer Variables

- int *iptr;
  - Data type of iptr is int *
  - iptr is to store the memory address of an integer variable
- float *fptr;
  - Data type of fptr is float *
  - fptr is to store the memory address of a floating point variable
- char *cptr;
  - Data type of cptr is char *
  - cptr is to store the memory address of a char variable

Storing Data and Memory Addresses

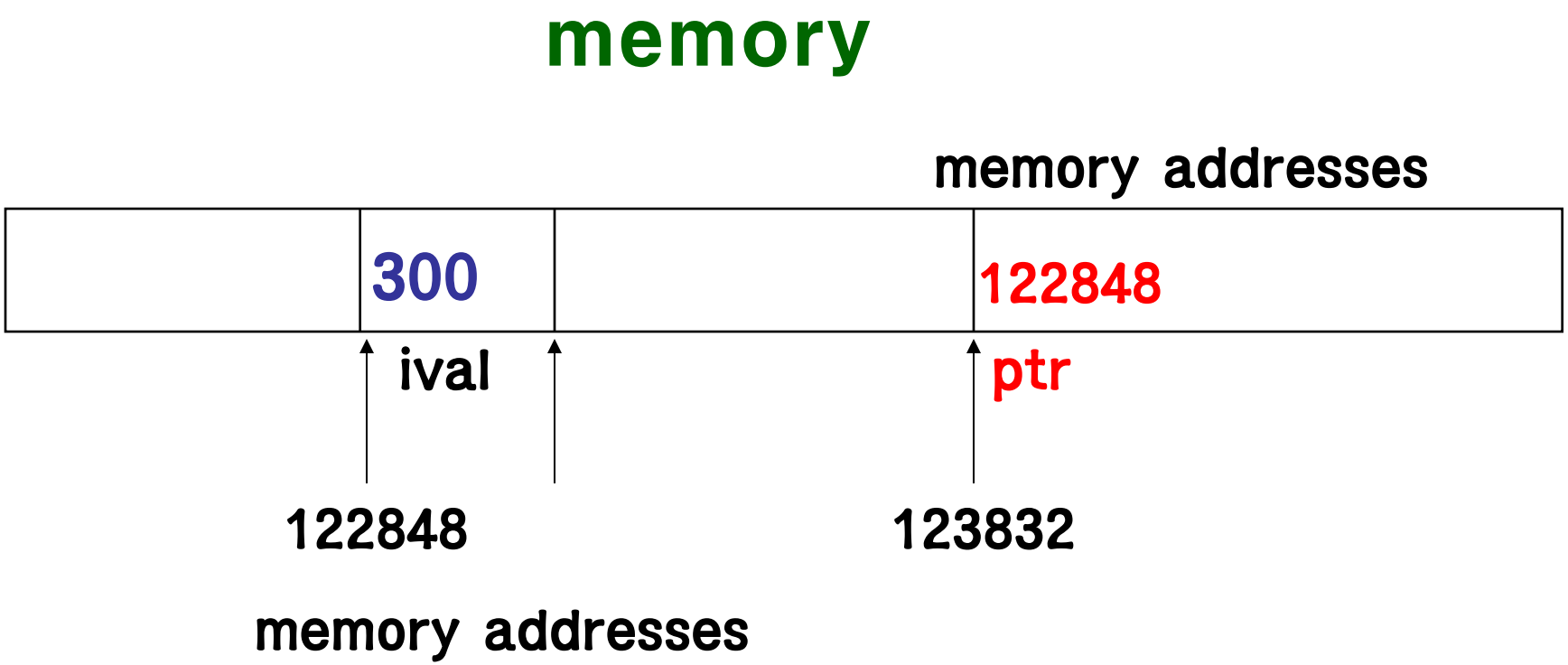


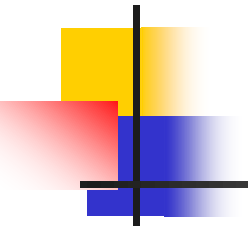


Address Operator

- Address Operator: &
- int ival, *ptr;
- ptr = &ival;
  - obtains the memory address of variable ival
  - stores it in ptr

Storing Data and Memory Addresses



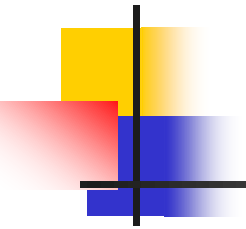


& Used with scanf

scanf (“%f”, &testval);

Indirection (Dereference) Operator *

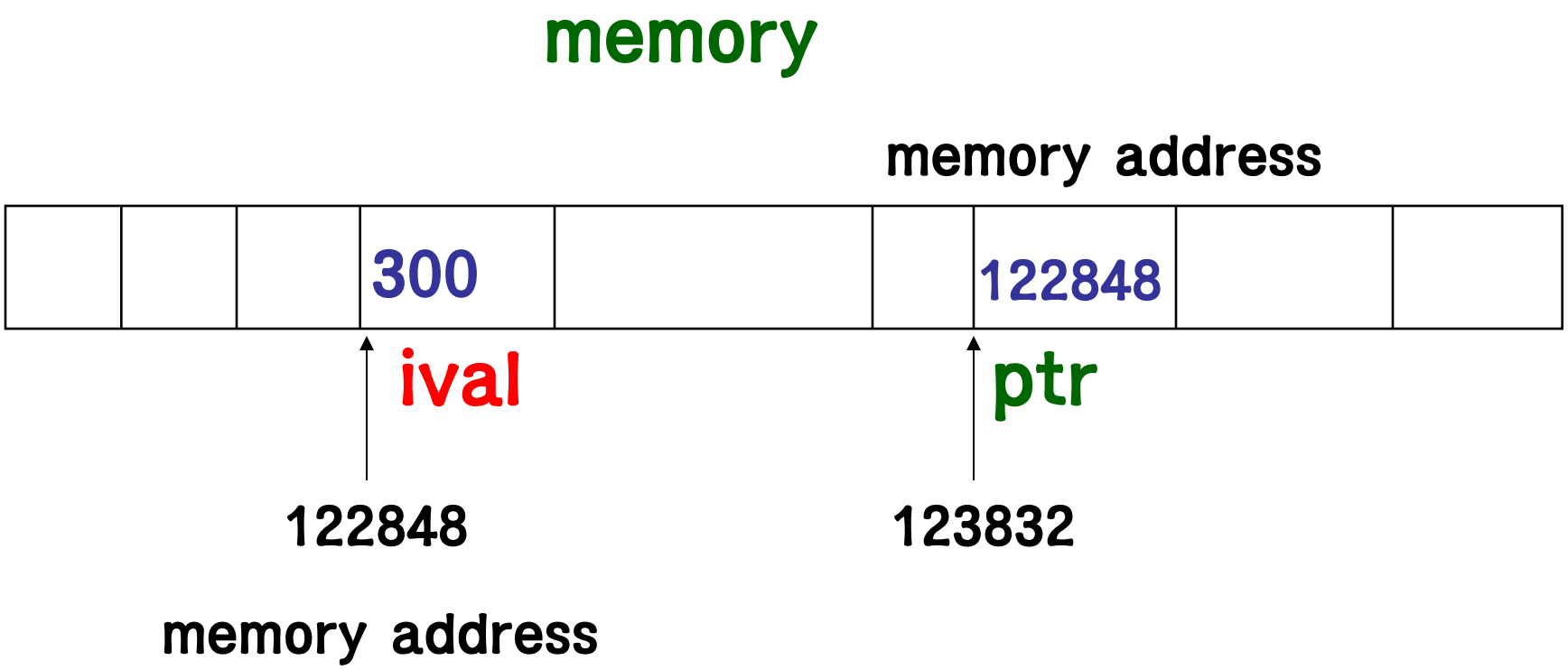
- Obtains the address stored in a pointer variable
  - bad notation
    - conflict with "multiply", and comments



Example

```
int  ival, newval, *ptr;
ptr = &ival;
*ptr = 300;
        /* gets the address stored in ptr (i.e., address of ival)
        and stores 300 in ival */
newval = *ptr * 27
       /* dereferences ival (reads the value stored in ival),
          multiplies by 27, and stores the new value in
          newval  */
```

Memory Diagram



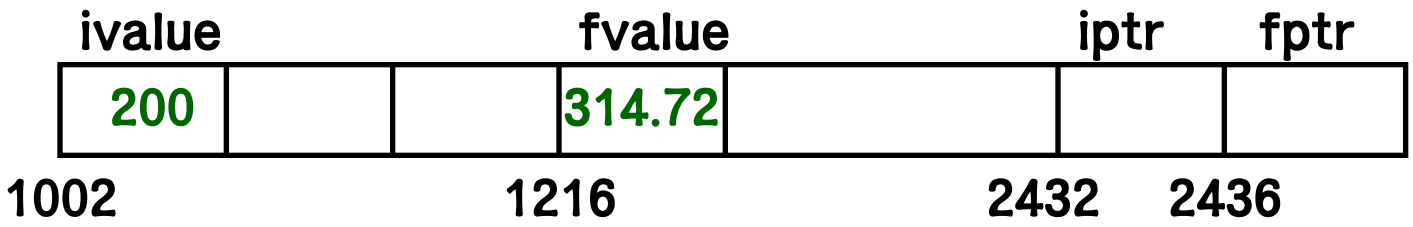
Exercise

```
float   *fptr;
int     *iptr;
float   fvalue;
int      ivalue;

ivalue = 200;
fvalue = 314.72;
iptr = &ivalue;         /* iptr has address of ivalue  */
fptr = &fvalue;        /* fptr has address of fvalue */

*fptr = *fptr – 300.0; /*    *fptr refers to fvalue   */
*iptr = *iptr + 300;    /*    *iptr refers to ivalue    */
```

Step by Step (1)

ivalue = 200;

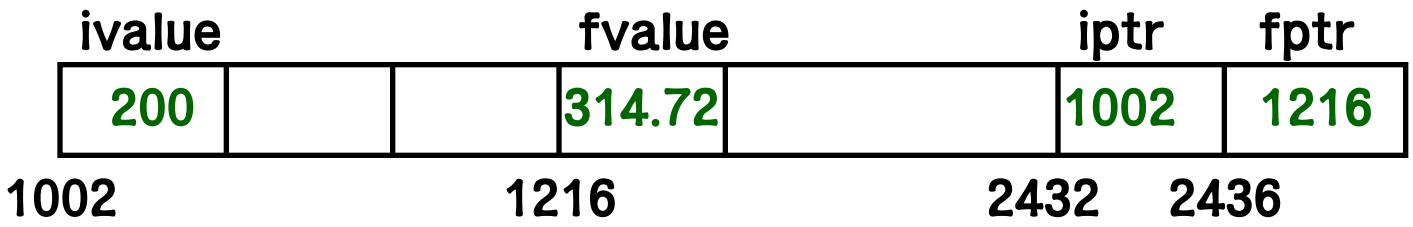
fvalue = 314.72;



Step by Step (2)

iptr = &ivalue;

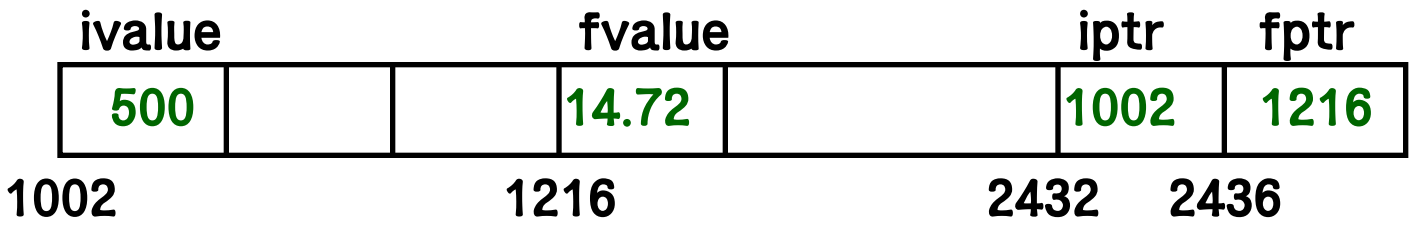
fptr = &fvalue;



Step by Step (3)

*fptr = *fptr – 300.0;

*iptr = *iptr + 300;



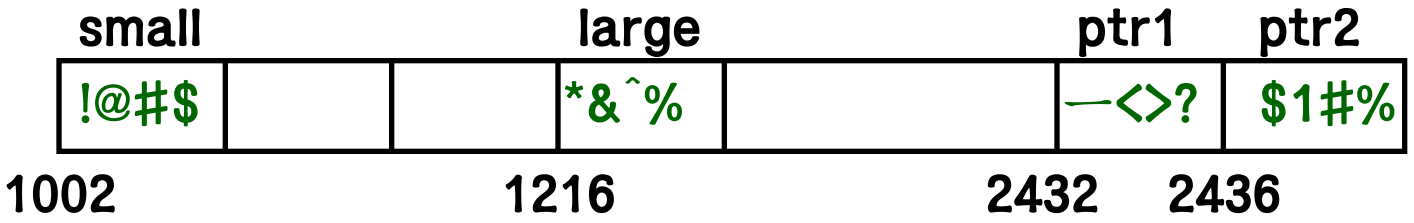
Exercise

In the following C program fragment, what are the values of “small” and “large” ?

```
int  *ptr1, *ptr2;
int   small, large;
small = 10;
large = 10000;
ptr1 = &small;
ptr2 = &large;
small = *ptr2;
large = *ptr1;
*ptr2 = 100;
```

Step by Step (1)

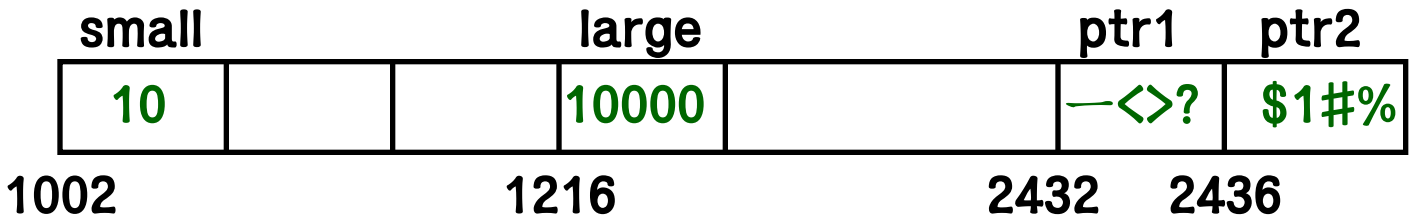
```
int  *ptr1, *ptr2;
int   small, large;
```



Step by Step (2)

small = 10;

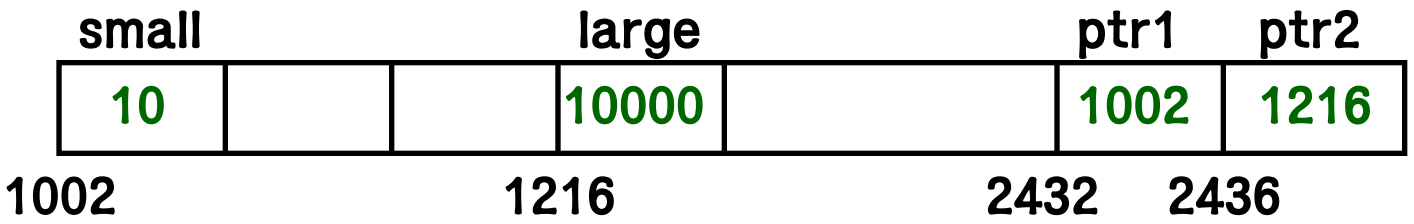
large = 10000;



Step by Step (3)

ptr1 = &small; ➔ ptr1 is the address of “small”

ptr2 = &large; ➔ ptr2 is the address of “large”



Step by Step (4)

small = 10;

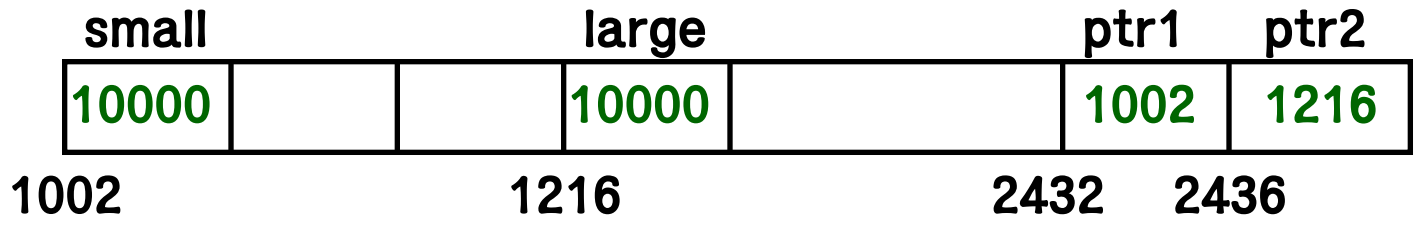
large = 10000;

ptr1 = &small; ➔ ptr1 is the address of “small”

ptr2 = &large; ➔ ptr2 is the address of “large”

small = *ptr2; ➔ small=10000, large=10000

large = *ptr1; ➔ large=10000, small=10000



Step by Step (5)

small = 10;

large = 10000;

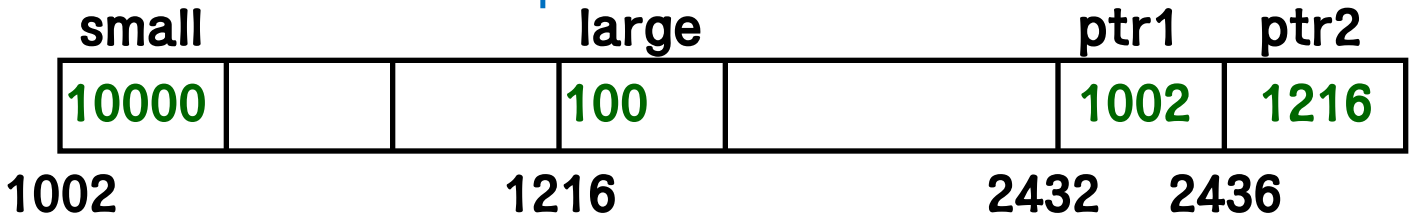
ptr1 = &small; ➔ ptr1 is the address of “small”

ptr2 = &large; ➔ ptr2 is the address of “large”

small = *ptr2; ➔ small=10000, large=10000

large = *ptr1; ➔ large=10000, small=10000

*ptr2 = 100; ➔ large=100, small=10000,
*ptr1=10000



Putting Them All Together

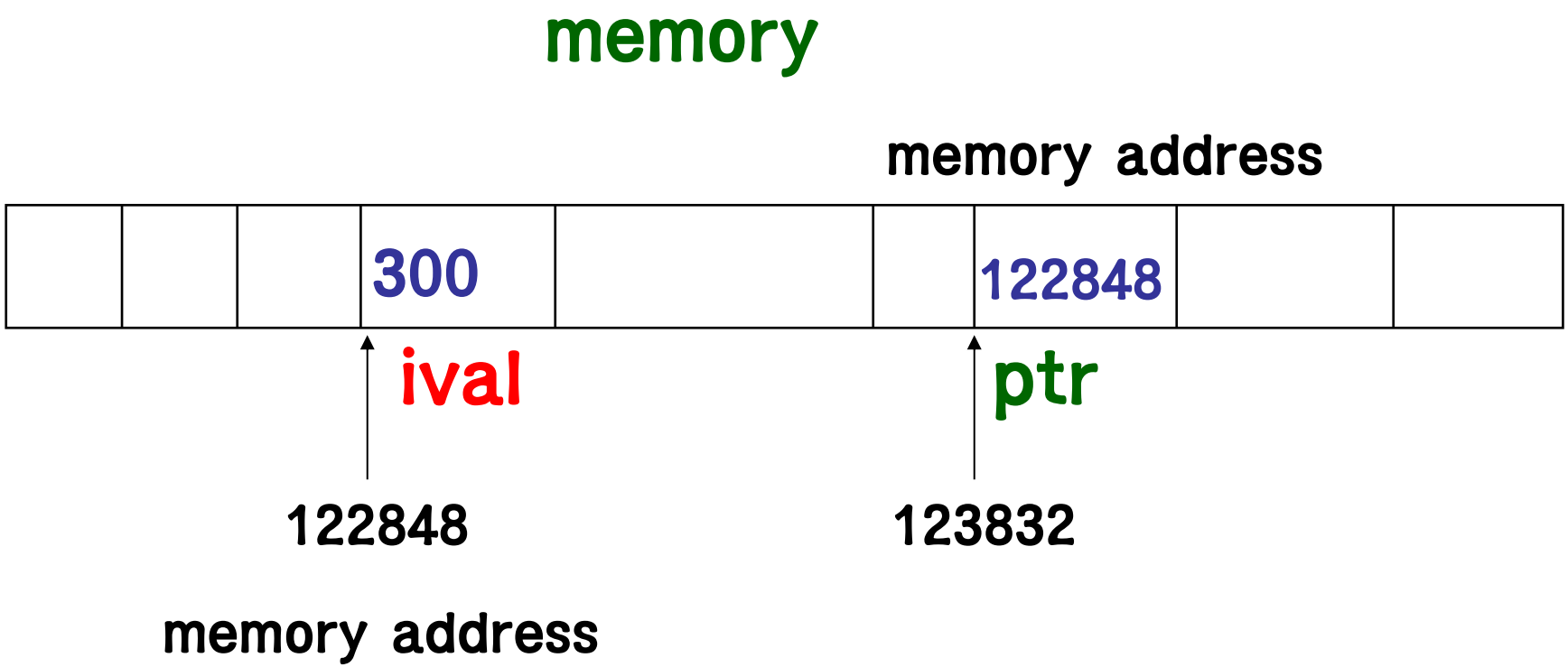
In the following C program fragment, what are the values of “small” and “large” ?

```
int  *ptr1, *ptr2;
int   small, large;
small = 10;
large = 10000;
ptr1 = &small;   ➔ ptr1 is the address of “small”
ptr2 = &large;   ➔ ptr2 is the address of “large”
small = *ptr2;   ➔ small=10000, large=10000
large = *ptr1;   ➔ large=10000, small=10000
*ptr2 = 100;     ➔ large=100, small=10000,
                    *ptr1=10000
```

Special Case

- * &ptr is the same as ptr

Memory Diagram



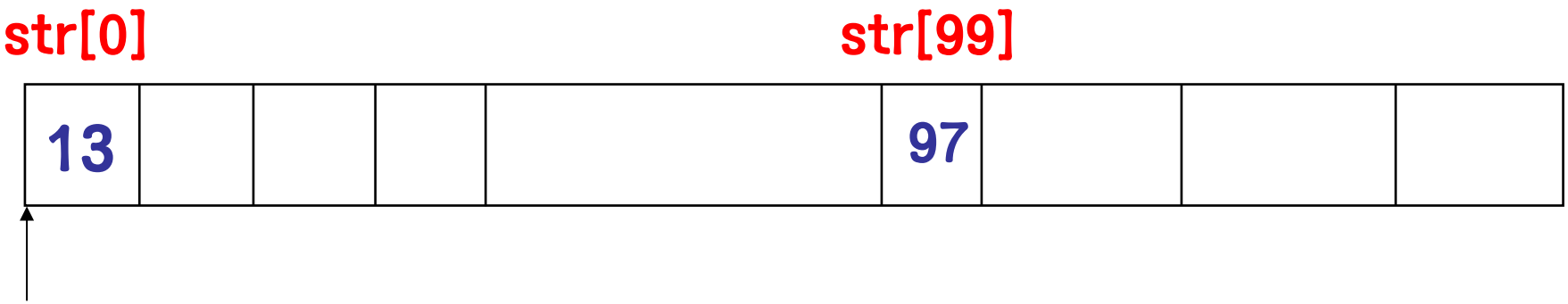
How It Works

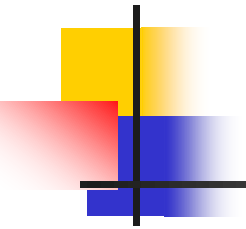
- &ptr is 123832
- * &ptr obtains the address stored in 123832
  (it is 122848)
  (it is the address stored in ptr)

Array and Pointer

- The name of an array is a constant pointer to the first element of the array.
- int str[100], num[1000];
  - str is a pointer to str[0] (str == &str[0])
  - num is a pointer to num[0] (num == &num[0])

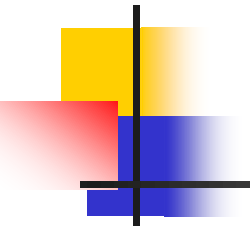
Memory Diagram





Pointer Arithmetic

- int str[100];
  - str+1 is a pointer to str[1]
    - ((str+1) == &str[0]+1 ==&str[1])
  - str+99 is a pointer to str[99]
    - ((str+99) == &str[0]+99 == &str[99])



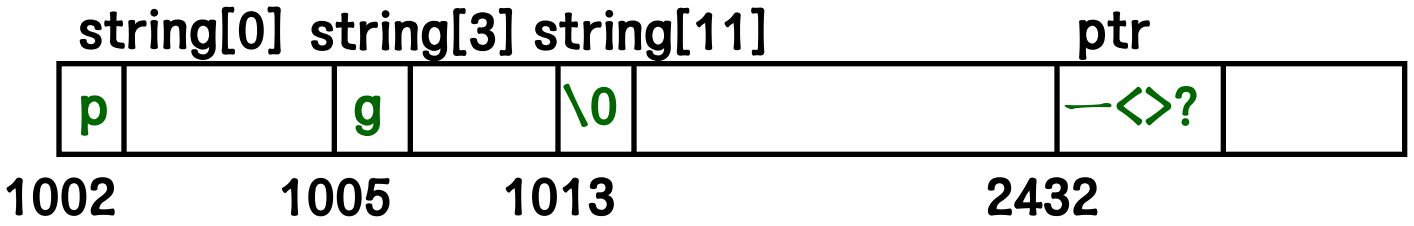
Exercise

In the following C program fragment, how do the values of ptr and string change?

```
char  *ptr, string[20] = "programming";
ptr = string;
ptr = ptr + 3;
*ptr = 'k';
```

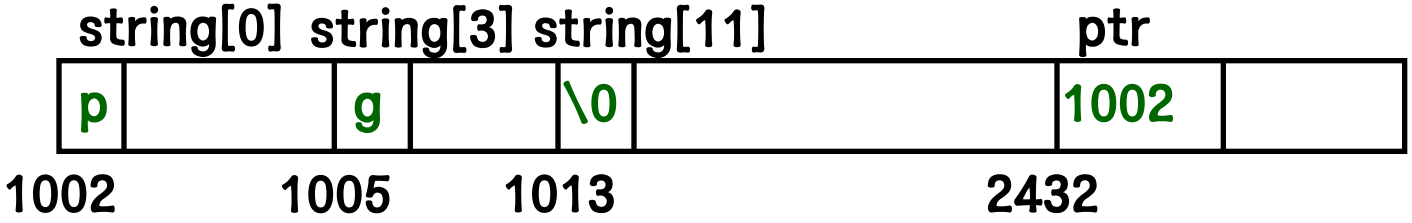
Step by Step (1)

char *ptr, string[20] = "programming";



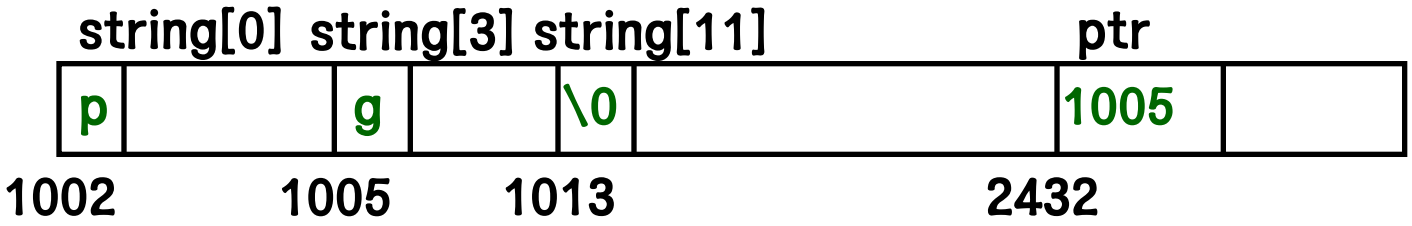
Step by Step (2)

ptr = string;



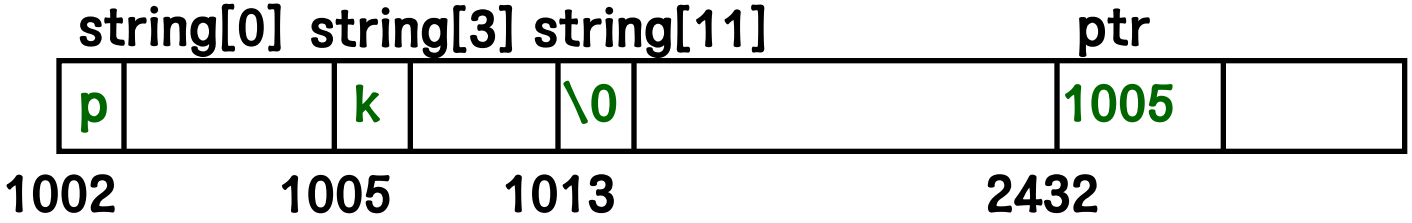
Step by Step (3)

ptr = **ptr+3;**



Step by Step (4)

*ptr = 'k';

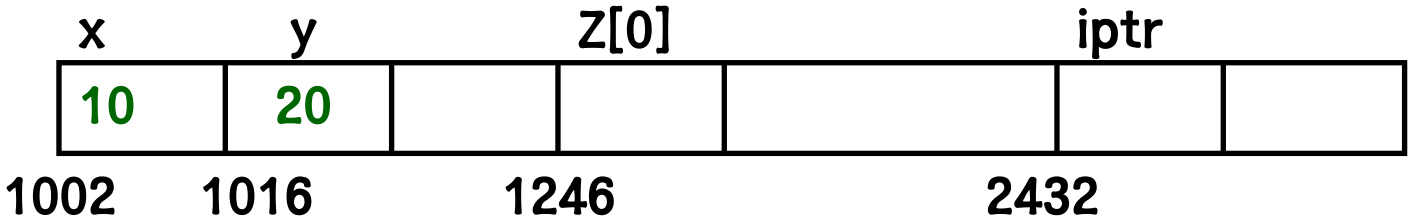


Exercise: (Draw a Memory Diagram.)

```
int  x = 10, y = 20, z [100];
int  *iptr;

iptr = &x;        /*   iptr  ?    */
y = *iptr;        /*   y  ?       */
*iptr = 100;      /*   x  ?       */
iptr  = &z[0];    /*   iptr  ?    */
```

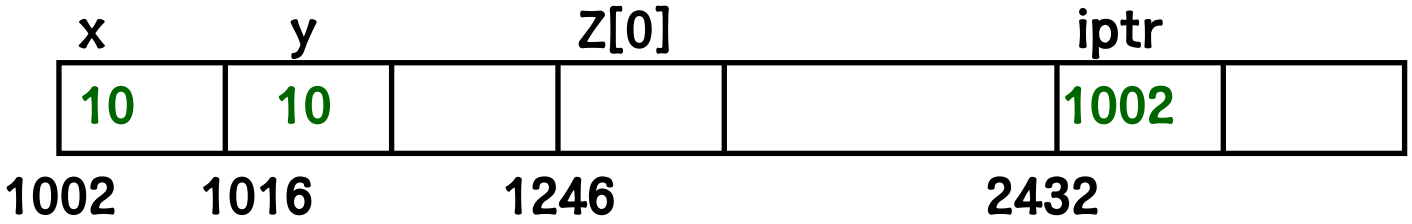
Step by Step (1)

- int x = 10, y = 20, z [100];
- int *iptr;



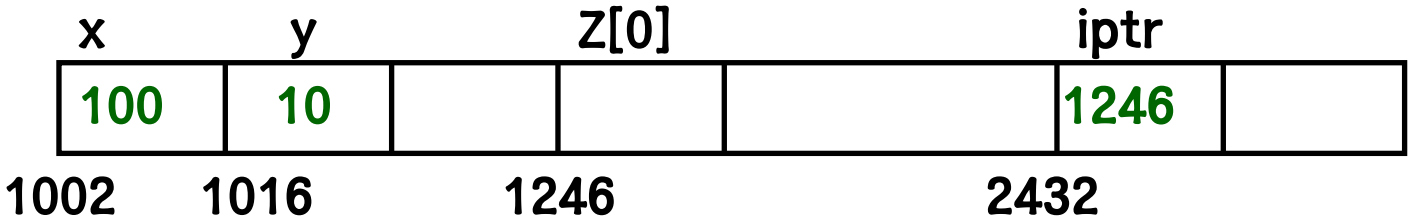
Step by Step (2)

- iptr = &x; /* iptr ? */
- y = *iptr; /* y ? */



Step by Step (3)

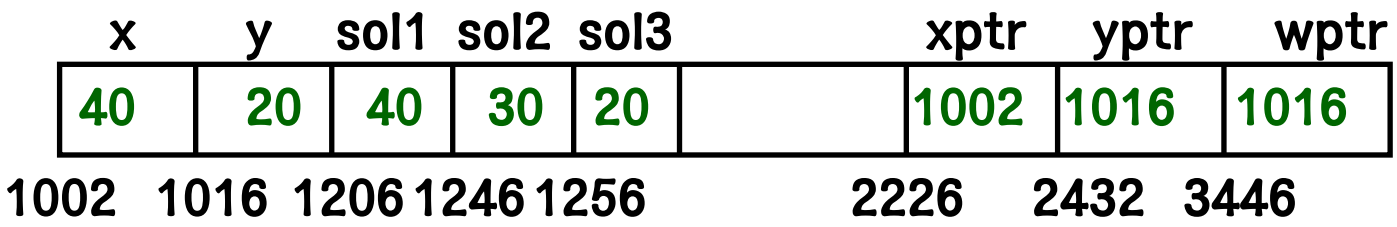
- *iptr = 100; /* x ? */
- iptr = &z[0]; /* iptr ? */



Exercise: values of sol1,sol2, sol3? (Draw a Memory Diagram.)

```
int  x = 40, y = 20, sol1, sol2, sol3;
int  *xptr, *yptr, *wptr;

xptr = &x;
yptr = &y;

sol1 = * * &xptr;
sol2 = 4 * *xptr / *yptr + 22;
sol3 = * (wptr = &y);
```

Solution


x
y
sol1
sol2
sol3
xptr
yptr
wptr
40
20
40
30
20
1002
1016
1016
1002
1016
1206
1246
1256
2226
2432
3446

Exercise: What Is the Output of the Program?

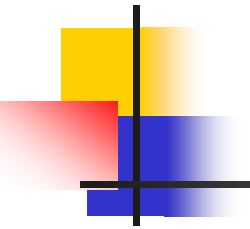
```
char *p;
char s[81]= “He drinks coke each day.”;
for (p=s+10; *p != ‘₩0’; p = p+1)
{
    if (*p == ‘c’)
        *p = ‘C’;
    if (*p == ‘d’)
        *p = ‘D’;
    if (*p == ‘e’)
        *p = ‘E’;
    if (*p == ‘ ’)
       *p = ‘₩n’;
}
printf (“%s₩n”, s);
```

Solution

- He drinks CokE
- EaCh
- Day.

Valid and Invalid Pointer Arithmetic

- char *p, s[100];
  - s[i] = *(s + i)
  - ++p is valid.
    - (increment the pointer before using it)
  - p++
    - (use the pointer and then increment it)
  - p = s is valid.
  - ++s is not valid. (cannot change a constant)
  - s = p is not valid. (cannot assign to a constant)

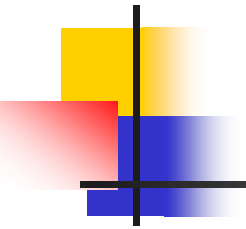


& and * as Unary Operators

- Normal Precedence and Associativity Rules
- Similar to -, ! Unary Operators
- Ex: 7 + *ptr1 / *ptr2

Some Illegal Constructs

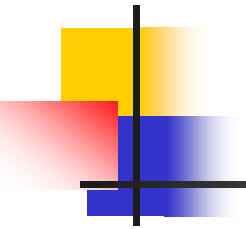
- &327
- &(i + 99)
- double dnum; int *ptr; ptr = &dnum;
  - data type mismatch



Need for Pointers

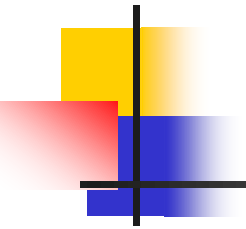
- Function Call by Reference
- Linking Data (linked list)
  - logical links
    - hot links on Web pages
  - physical link for speed

Function
(Call by Reference)



Call by Reference (Address)

- Passes an address as an argument to a function
  - address = pointer to memory address
  - The argument must NOT be an "expression to be evaluated"
- Avoids copying data



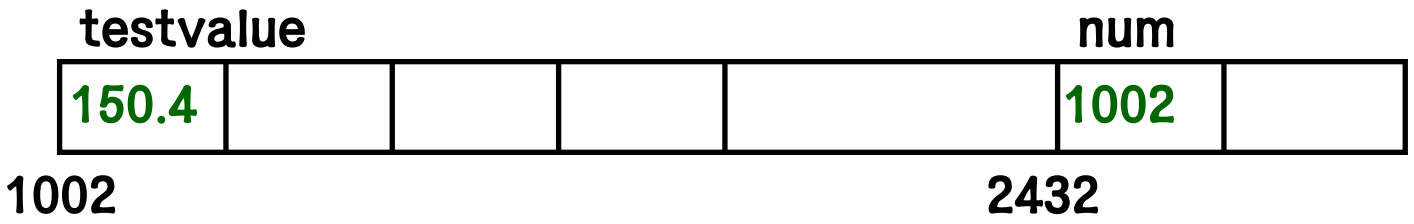
Call by Reference

- The called function can change the arguments in the calling function
  - side effects, similar to global variables
- Multiple address arguments can result in changes in multiple input values.
  - way to return multiple values
  - (call by value can return only one data.)

Example

```
void newVal (float *);  /* function prototype */

void main ()
{
   float  testval;

   printf ("₩nEnter a number: ");
   scanf ("%f", &testval);

   newVal (&testval);       /* function call        */

}

void newVal (float *num) /* function definition */
{
   *num = *num + 20.2;
}
```

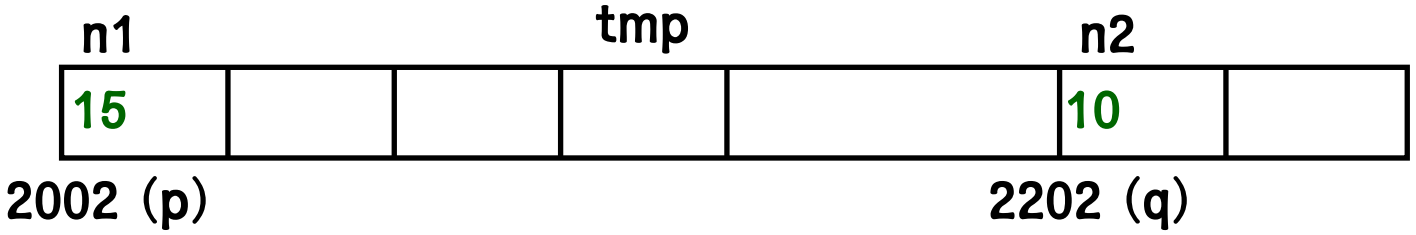
Memory Diagram



Exercise

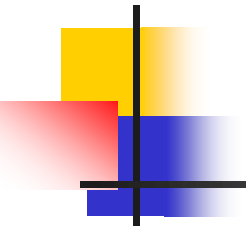
```
void swap (int *, int *);  /* function prototype */

void main ()
{
   int n1, n2;

   /* read n1 and n2    */

   swap (&n1, &n2);       /* function call        */
}

void swap (int *p, int *q) /* function definition */
{
   int  tmp;
   tmp = *p;
   *p = *q;
   *q = tmp;
}
```

Memory Diagram



Example: Function "Returning" Multiple Values

```
void newVal (int, int, float *, float *);

void main()
{
   int n1, n2;
   float f1, f2;
   /*  read n1 and n2  */
   newVal (n1, n2, &f1, &f2);
}

void newVal (int val1, int val2,
             float *sum, float *remain)
{
   *sum = val1 + val2;
   *remain = val1 % val2;
}

/* "returns" two values */
/* side effects – same as global variables */
```

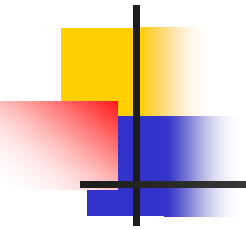


Exercise

Write a function void minmax with 4 parameters.

minmax has 2 parameters whose data types are int and 2 parameters whose data types are int *.

minmax receives 2 integers, and determines the smallest and largest of the 2 integers, and returns them through the two pointer parameters.



Solution

```
void minmax (int num1, int num2,
                int *min, int *max)
{
    if (num1 <= num2) {
      *min = num1;
      *max = num2;
    }
    else {
       *min = num2;
       *max = num1;
    }
}
```

Exercise

- Write a C (main) program that reads two integers, N1 and N2, and calls a function, calculate, by passing the addresses of N1 and N2.
- calculate calculates N1*N2, N1/N2, N1%N2, N1+N2 and returns the four results. The main program prints the results.
- (* hint: calculate has six parameters, (int *n1, int *n2, int *re1, int *re2, int *re3, int *re4), where re1, re2, re3, re4 store the four results.

Exercise

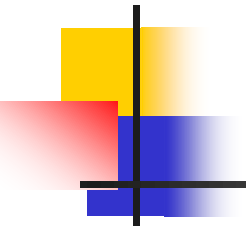
Write a C program that computes and prints the total salaries of people.
The program receives 5 base salaries (float) and 5 overtime payments (float), stores them in arrays base and overtime. The program calls a function void totpay with 3 array arguments, base, overtime and total.

totpay computes the total pay for each person by adding the base salary and overtime payment, and storing the result in total.

The main program prints total.

Solution (Sketch)

```
void totpay (float[5], float[5], float[5]);

void main()
{
   /*  declarations   */
   float base[5], overtime[5], total[5];
   /*  read and store values in array pay */
   totpay (base, overtime, total);
   /*  print total    */

void totpay (float b[], float o[], float t[])
{
  /* compute and store total for each person  */
}
```

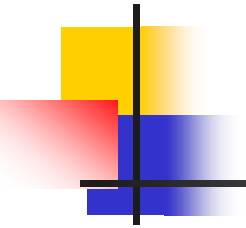


Homework

- Must follow the problem solving steps (including the drawing of memory diagrams).

1, 2

- 1. Write a C function named change() that accepts a single-precision number and the addresses of the integer variables named *quarters, dimes, nickels,* and *pennies*. The function should determine the number of quarters, dimes, nickels, and pennies in the number passed to it and write these values directly into the respective variables declared in its calling function.
- 2a. Write a function named secs() that accepts the time in hours, minutes, and seconds; and determines the total number of seconds in the passed data. Write this function so that the total number of seconds is returned by the function as an integer number.
- 2b. Repeat Exercise 2a but also pass the address of the variable *totSec* to the function secs(). Using this passed address, have secs() directly alter the value of *totSec*.

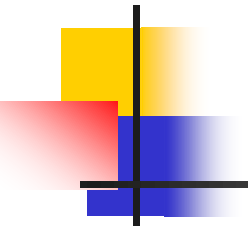


3

- 3. Write a C function named liquid() that is to accept an integer number and the addresses of the variables gallons, quarts, pints, and cups. The passed integer represents the total number of cups, and the function is to determine the number of gallons, quarts, pints, and cups in the passed value. Using the passed addresses, the function should directly alter the respective variables in the calling function. Use the relationships of 2 cups to a pint, 4 cups to a quart, and 16 cups to a gallon.

4, 5

- 4. Write a program that has a declaration in main() to store the following numbers into an array named *rates*: 6.5, 8.2, 8.5, 8.3, 8.6, 9.4, 9.6, 9.8, 10.0. There should be a function call to show() that accepts the *rates* array as a parameter named *rates* and then displays the numbers in the array.
- 5. Write a program that declares three one-dimensional arrays named *price, quantity*, and *amount*. Each array should be declared in main() and should be capable of holding 10 double-precision numbers. The numbers that should be stored in *price* are 10.62, 14.89, 13.21, 16.55, 18.62, 9.47, 6.58, 18.32, 12.15, 3.98. The numbers that should be stored in *quantity* are 4, 8.5, 6, 8.35, 9, 15.3, 3, 5.4, 2.9, 4.8. Your program should pass these three arrays to a function called extend(), which should calculate the elements in the *amount* array as the product of the equivalent elements in the *price* and *quantity* arrays (for example, amount[1] = price[1] * quantity[1]). After extend() has put values in the *amount* array, the values in the array should be displayed from within main().



End of Lecture