CENG 10003 - PROGRAMMING
POINTERS

Peter Basl, PhD

Outline

- Pointers
- Declaration & Initialization
- Pointers & Functions
- Passing Arrays to Functions

Pointers

- **Pointers** – variables that most often contain a memory address as their value. The memory address points to another variable
- Used to work with variables, functions, and data structures through their memory addresses

Declaring Pointer Variables

- As usual, pointer variables must be declared before they can be used

```
int x = 0;
int age = 30;
int *ptrAge; // a pointer to an integer data type
```

Indirection Operator (*)
An operator placed in front of the variable name to declare a pointer

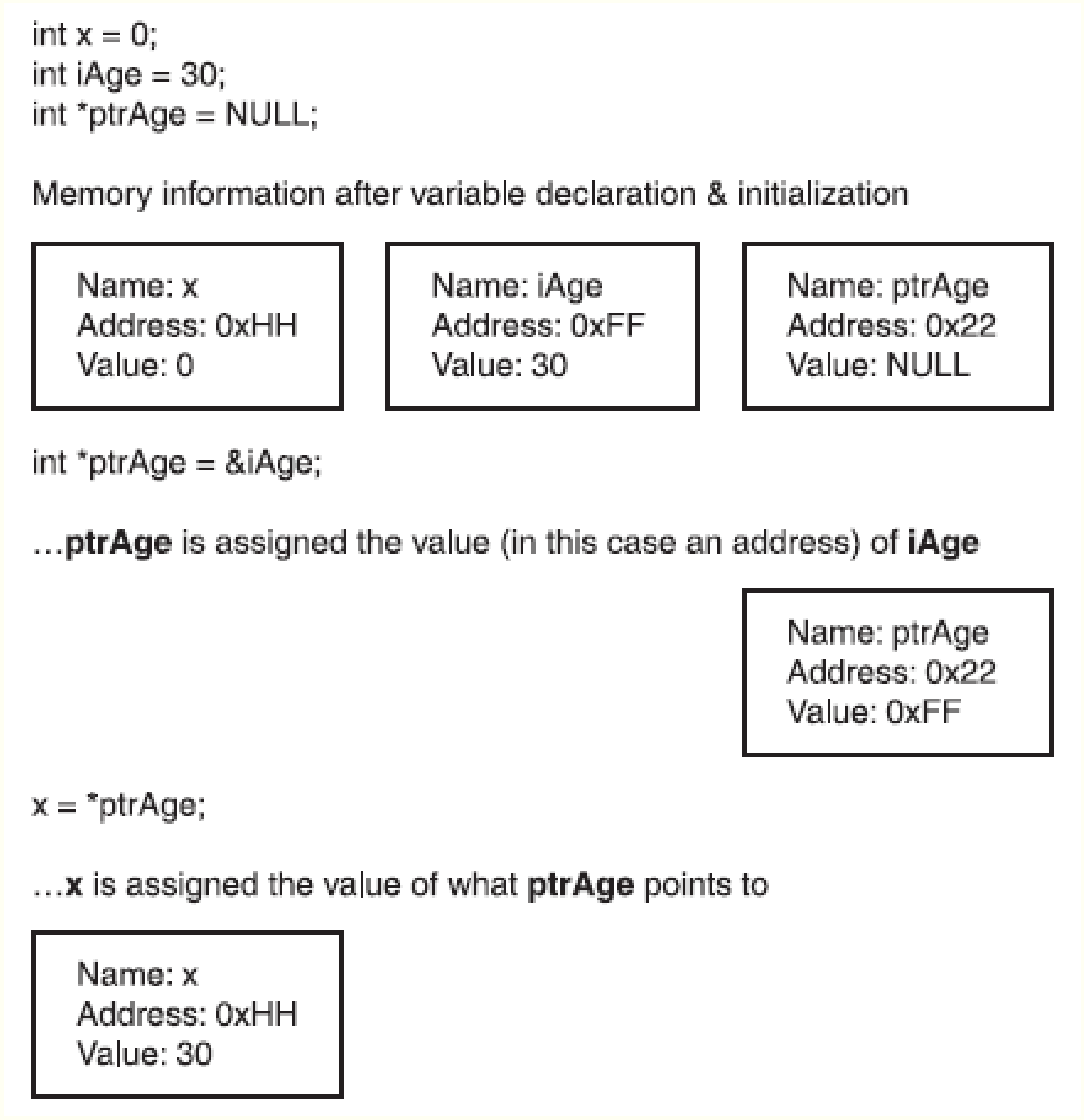
Initializing Pointer Variables

- To indirectly reference a value through a pointer, assign the address of the variable storing this value to the pointer

```
int x = 0;
int age = 30;
int *ptrAge; // a pointer to an integer data type
ptrAge = &age;
x = *ptrAge; // referencing value through the pointer
```

Address Operator (&)

An operator used to determine the memory address of a variable



```
int x = 0;
int iAge = 30;
int *ptrAge = NULL;
```

Memory information after variable declaration & initialization

Name: x	Name: iAge	Name: ptrAge
Address: 0xHH	Address: 0xFF	Address: 0x22
Value: 0	Value: 30	Value: NULL

```
int *ptrAge = &iAge;
```

…**ptrAge** is assigned the value (in this case an address) of **iAge**

Name: ptrAge
Address: 0x22
Value: 0xFF

```
x = *ptrAge;
```

…**x** is assigned the value of what **ptrAge** points to

Name: x
Address: 0xHH
Value: 30

Initializing Pointer Variables

- Pointer variables are either initialized with another variable's memory address, with 0, or with the keyword `NULL`

```
int *ptr1;
int *ptr2;
int *ptr3;

ptr1 = &x;
ptr2 = 0;
ptr3 = NULL;

ptr1 = 5; // wrong
ptr1 = x; // wrong

*ptr1 = 5; // correct (will change x value to 5)
```

Printing Pointer Variable Contents

- Use the `%p` conversion specifier to print the memory address of pointers and non-pointer variables

```
int x = 1;
int *ptr = &x;
*ptr = 5;

printf(“*ptr = %p\n&x = %p\n”, ptr, &x);
```

- To print the actual values, you can use the following instead

```
printf(“*ptr = %d\n&x = %d\n”, *ptr, x);
```

Passing by Value

- In functions we used so far, we always passed arguments by value
- This involves making a copy of the incoming argument for the function to use
- This means two things:
  - It increases the amount of memory used by the program. This is not significant, unless you have limited memory in your device
  - The function is unable to modify the original contents of the incoming parameters. This is not bad; if you want to protect the original values

Passing by Reference

- **Passing by Reference** – means passing the address of the variable (argument) to the function using indirection
- It's an important use for pointers
- It results in the following:
  - Less use of memory
  - Allows the function to modify the original value of the variable (can be dangerous if not used carefully & correctly)

Passing by Reference

```
void demoPassByReference(int *);

main() {
int x = 0;
printf(“Enter a number: “);
scanf(“%d”, &x);
demoPassByReference(&x); // pass address of variable
printf(“x = %d”, x); }

void demoPassByReference(int *ptrX) // parameters is a pointer
{
printf(“Original value of X = %d\n”, *ptrX);
*ptrX += 5;
}
```

Functions with Multiple Outputs

- Passing by reference allows having a function that calculates more than one output with no or just the one `return` statement

```
void squareCalc(int, int *, int *); // prototype

squareCalc(side, &perimeter, &area); // function call

void squareCalc(int s, int *p, int *a) // function implementation
{
*p = 4*s;
*a = s*s; }
```

Arrays & Pointers

- Passing an array name to a pointer assigns the first memory location of the array to the pointer variable
- Simply, the array name contains a memory address pointing to its first element & can be treated like a pointer

```
int numbers[5] = [1,2,3,4,5];
int *ptr = numbers;

printf(“Address of points: %p\n”, ptr);
printf(“Address of first array element: %p\n”, &numbers[0]);

printf(“Pointer points to: %d\n”, *ptr);
printf(“First element of array: %d\n”, numbers[0]);
```

Passing Arrays to Functions

- Because array names are actually pointers, it’s not necessary to deal with address and indirection operators when passing arrays to functions
- **Arrays passed as arguments are automatically passed by reference**
- To pass an array to a function, the prototype and header should be defined so that they expect to receive an array as an argument in addition to another parameter for the size of the array (particularly when passing numerical arrays)

Passing Char Arrays to Functions

```
int nameLength(char []);
main() {
char name[20] = {‘\0’};
printf(“Enter your first name: “);
scanf(“%s”, name);
printf(“Your first name contains “);
printf(“%d characters”, nameLength(name)); }

int nameLength(char name[]) {
int x = 0;
while (name[x] != ‘\0’)
x++;
return x; }
```

Passing Numerical Arrays to Functions

```
#define SIZE 3
int squareNumbers(int [], int);
main() {
int x, numbers[SIZE] = {2, 4, 6};
printf(“The current array values are: “);
for(x=0; x<SIZE;x++)
printf(“%d “, numbers[x]);
squareNumbers(numbers, SIZE);
printf(“The modified array values are: “);
for(x=0; x<SIZE;x++)
printf(“%d “, numbers[x]);}

int squareNumbers(int num[], int size) {
for(int x=0; x<size, x++)
num[x] = num[x]*num[x]; }
```