CPE102 Programming II

Week 4

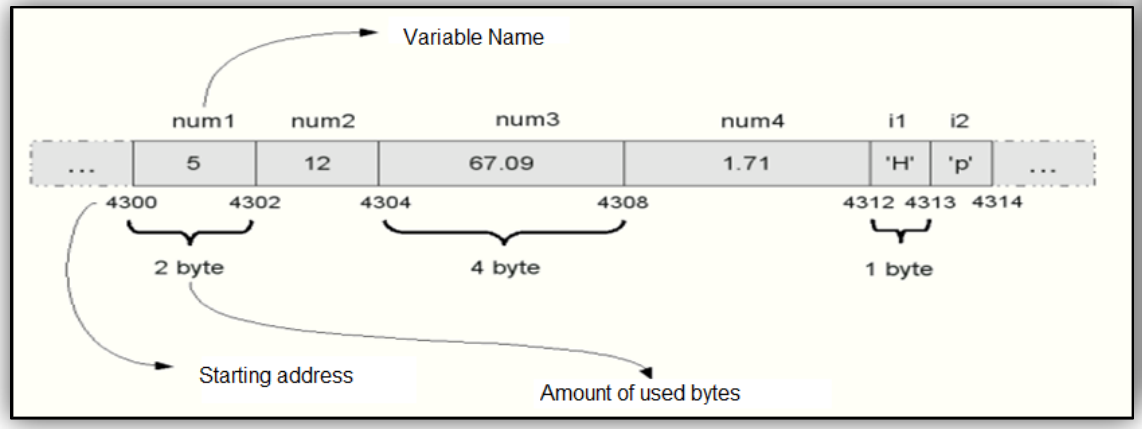
Pointers

Dr. Nehad Ramaha,
Computer Engineering Department
Karabük Universities

Memory Structure

- When a variable defined it is stored somewhere in memory.
- Memory can be thought as block consist of cells.
- Number of cells that will be reserved in the memory for the variable depends on its type.

Memory Structure



ere | MemoryStr.c

```
1  #include <stdio.h>
2  int main (void) {
3       // Variables are defined:
4       int num1, num2;
5       float num3, num4;
6       char i1, i2;
7       // Assignment to variables:
8       num1 = 5;
9       num2 = 12;
10      num3 = 67.09;
11      num4 = 1.71;
12      i1 = 'H';
13      i2 = 'p';
14      return 0;  }
```

- When a variable is defined, a space required for the variable is reserved in the memory.
- E.g. definition int num1 reserves 2 byte space.
- After that if the value 5 is assigned on variable num1, 5 is stored in memory location allocated for that variable.
- Actually, all operations taken on variable num1 is the modification of cells in the memory location between 4300 and 4302.
- Variable is actually a memory location reserved for a particular label.

What are Pointers?

- Pointers are variables that store memory addresses which are memory location of other variables.
  - dataType *pointerVar;
  - int *ptr;
  - char *ptr;
- Pay attention to define pointer to be suitable for the data type it points.
- A float variable must only be pointed by a float type pointer.

Use address and value at address operators

- We use the address (& operator) to get the address of a variable.
- We use the value at the address (* operator) to get the value stored at any given address of a variable.
- Note: In the following code &num gives us the address of variable num and *(&num) gives us the value stored at the address of the variable num.

Output:
Value in num: 10
Address of num: 6422044
Value at the address of num: 10

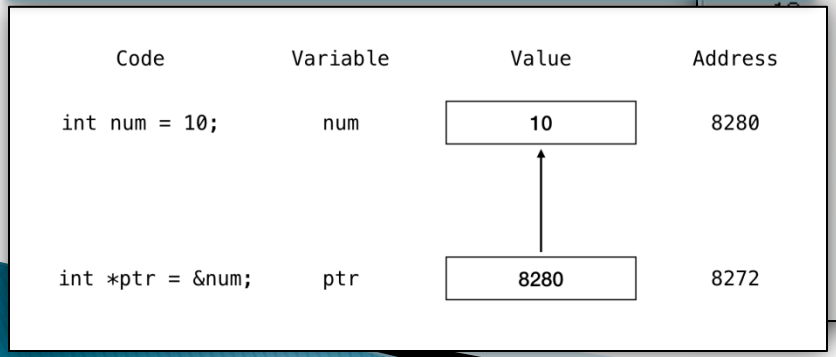
*PrintingPtr.c

```c
#include <stdio.h>

int main(void) {
  // creating variable num
  int num;
  // assigning value to num
  num = 10;
  // printing the value stored in variable num
  printf("Value in num: %d\n", num);
  // printing the address of the variable num
  printf("Address of num: %ld\n", &num);
  // printing the value stored at the address of the variable num
  printf("Value at the address of num: %d\n", *(&num));
  return 0;}
```

Initializing pointer variable with address

- To initialize pointer variable with the address of another variable we use the & address of operator.

Output:
Value stored in num: 10
Address of num variable: 8280
Address of ptr variable: 8272
Value stored in num using ptr variable: 10



Ptr.c

```c
#include <stdio.h>
int main(void) {

    int num = 10;    // num variable
    int *ptr;        // pointer variable

    // assign address of num variable to ptr pointer variable
    ptr = &num;

    // print the value stored in num using the num variable
    printf("Value stored in num: %d\n", num);
    // print address of num stored in ptr variable
    printf("Address of num variable: %ld\n", ptr);
    // print address of ptr variable
    printf("Address of ptr variable: %ld\n", &ptr);
    // print the value stored in num using ptr variable
    printf("Value stored in num using ptr variable: %d\n", *ptr);

    return 0;}
```

Updating the value of a variable via pointer

- Using pointers, we can change the values of stored variables.
- For accessing or changing the value of a variable with pointer, we should use * character in the beginning of pointer name

Output:
num: 10
Updating value of num via ptr...
num: 20
num via ptr: 20

UpdatingVariablePtr.c

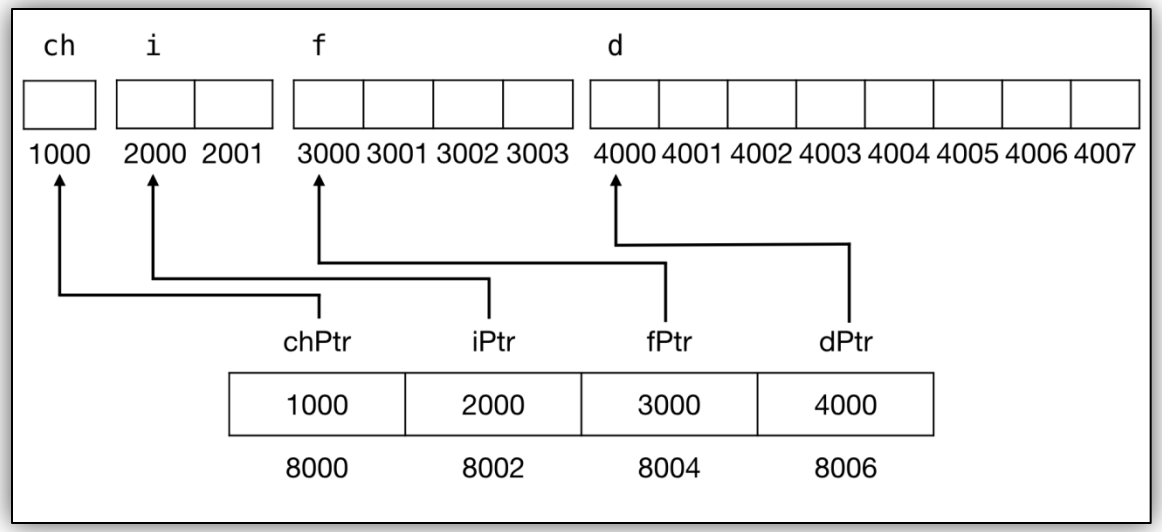
```c
#include <stdio.h>
int main(void) {

    int num = 10;       // num variable
    int *ptr = NULL;    // ptr pointer variable

    ptr = &num;         // assigning the address of num to ptr

    // printing the value of num - Output: 10
    printf("num: %d\n", num);
    printf("Updating value of num via ptr...\n");
    *ptr = 20;          // updating the value of num via ptr

    // printing the new value of num - Output: 20
    printf("num: %d\n", num);
    printf("num via ptr: %d\n", *ptr);

    return 0; }
```

Size of pointer variables

- Pointer variables stores the address of other variables. And these addresses are **integer value**.
- We can use the sizeof() operator to find the size of the pointer variable.
- The size of the pointer variables depends on the compiler. For example, Borland C/C++ compiler takes 2 bytes to save integer value so, pointer size will be 2 bytes. Whereas, Visual C++ compiler takes 4 bytes to save integer values so, in that case size of the pointer will be 4 bytes.

Pointers for the variables

- In the following image we are assuming that an integer value takes 2 bytes. So, pointers variable size is 2 bytes.
  - char *chPtr = &ch;
  - int *iPtr = &i;
  - float *fPtr = &f;
  - double *dPtr = &d;



Size of pointer – Example

SizeofPtr.c

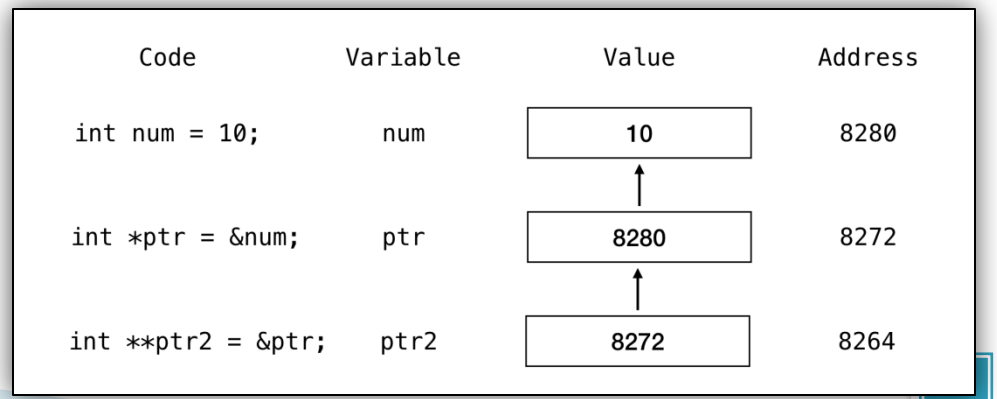
```c
#include <stdio.h>
int main(void) {
  // variables
  char ch = 'a';
  int i = 10;
  float f = 12.34;
  double d = 12.3456;
  // pointers
  char *chPtr = &ch;
  int *iPtr = &i;
  float *fPtr = &f;
  double *dPtr = &d;
  // print value
  printf("ch: %c\n", *chPtr);
  printf("i: %d\n", *iPtr);
  printf("f: %f\n", *fPtr);
  printf("d: %lf\n", *dPtr);
  // print address
  printf("\n");
  printf("Address ch: %ld\tsizeof ch: %ld\n", &ch, sizeof(ch));
  printf("Address i: %ld\tsizeof i: %ld\n", &i, sizeof(i));
  printf("Address f: %ld\tsizeof f: %ld\n", &f, sizeof(f));
  printf("Address d: %ld\tsizeof d: %ld\n", &d, sizeof(d));
  printf("\n");
  printf("\tsizeof chPtr: %ld\n",sizeof(chPtr));
  printf("\tsizeof iPtr: %ld\n", sizeof(iPtr));
  printf("\tsizeof fPtr: %ld\n", sizeof(fPtr));
  printf("\tsizeof dPtr: %ld\n", sizeof(dPtr));
  return 0;}
```

```
ch: a
i: 10
f: 12.340000
d: 12.345600

Address ch: 6422015 sizeof ch: 1
Address i: 6422008     sizeof i: 4
Address f: 6422004     sizeof f: 4
Address d: 6421992     sizeof d: 8

        sizeof chPtr: 8
        sizeof iPtr: 8
        sizeof fPtr: 8
        sizeof dPtr: 8
```

The compiler I am using is using 8 bytes to store integer values for pointer variables.

Pointers that point other Pointers(Pointers Chaining)

- As seen that pointers store the memory addresses of variables.
- Pointer is also a variable and another pointer can point to it.
- If we define a pointer variable that points another pointer; we use '**' in the beginning of pointer name.
- Number of * can change. If we define a pointer that points an other pointer that points an other pointer we have to use '***' (3 pointers chain).
- To create a second pointer variable to point at the first pointer variable we use the following syntax.
  - dataType **secondPtr = &firstPtr;



Update value of integer variable via second integer pointer variable

*ChainPtr.c

```c
#include <stdio.h>

int main(void) {

  int num = 10;// num variable
  int *ptr = NULL;// ptr pointer variable
  int **ptr2 = NULL;// second ptr2 pointer variable

  ptr = &num;// assigning the address of num to ptr
  ptr2 = &ptr;  // assigning the address of ptr to ptr2

  // printing the value of num - Output: 10
  printf("num: %d\n", num);
  printf("num via ptr: %d\n", *ptr);
  printf("num via ptr2: %d\n", *(*ptr2));

  // updating the value of num via ptr2
  printf("Updating value of num via ptr2...\n");
  *(*ptr2) = 20; // So, *ptr2 is pointing at ptr and *ptr is pointing at num.

  // printing the new value of num - Output: 20
  printf("num: %d\n", num);
  printf("num via ptr: %d\n", *ptr);
  printf("num via ptr2: %d\n", *(*ptr2));
  return 0;}
```

```
num: 10
num via ptr: 10
num via ptr2: 10
Updating value of num via ptr2...
num: 20
num via ptr: 20
num via ptr2: 20
```

Pointer Arithmetic

- We can use increment (++), decrement (--), addition (+) or subtraction (-) operators with pointers. But this value must be integer.
- When we increment the pointer by 1, pointer shows the next data block.
- New pointer value depends on the data type that pointer shows.
  - int i, *iPtr;
  - iPtr= &i; // Assume iPtr shows address 1000
  - iPtr+= 2 // After this operation new value of iPtr is 1004, assuming that the compiler using 2 bytes to store integer values for pointer.

Pointer Arithmetic

- Assuming that the compiler using 4 bytes to store integer values for pointer.
- int i=10, *iPtr;
- iPtr= &i; // Assume iPtr shows address 1000
- (*iPtr) ++; // Causes to increment value stored in the address 1000. (i++)
- iPtr++; // Causes iPtr to show address 1004 in memory
- (*iPtr) +=2; // Increase value stored in 1000 by 2. (i+=2)
- (*iPtr) =7; // Assign 7 in address 1000. (i=7)
- *(iPtr+2) = 5; // Assign 5 in address 1008.

Thanks ☺