CSCI 1380

May 1, 2017

PROGRAM MEMORY

When a program runs, relevant data is stored in the computer's **memory**.

Memory includes CPU cache, RAM, hard drives, and network.

A program does not “see” these different types of memory; every place in memory is just another location, called an **address**.

ADDRESSES

A computer’s memory is just a very long sequence of addresses that start at 0x0 (0 in “address-speak”, i.e., hexadecimal) and end at a very large number.

To get the address of a variable, use the & operator.
For example:

```
void printingAddresses()
{
  int n = 7;
  cout << “The address of n is: ” << &n;

  float f = 3.14;
  cout << “\nThe address of f is: ” << &f;
}
```

To get the address of a variable, use the & operator.
For example:

```
void sumThreeNums(int x, int y, int z, int &res)
{
  res = x + y + z;
}
```

POINTERS

If `n` is an `int`, then what is the type of `&n`?

(It should be a type that denotes an address of an `int`.)

Variables that store addresses are called **pointers**.

The syntax for pointers is **vartype***.
For example:

```
void declaringPointers()
{
  int n = 7;
  int* np = &n;
  cout << “The address of n is: ” << np;

  float f = 3.14;
  float* fp = &f;
  cout << “\nThe address of f is: ” << fp;
}
```

DEREFERENCING POINTERS

If an address is stored in a pointer, the value stored in the address can be obtained by **dereferencing** the pointer.

Adding an asterisk before the pointer’s name gives an expression that acts like the variable.

```
void dereferencingPointers()
{
   int n = 7;
   int* p = &n;

   cout << “The address of n is: ” << p;
   cout << “\nThe value of n is: ” << n;
   cout << “\nThe value of n is: ” << *p;

   n = 8;
   cout << “\nThe value of n is: ” << *p;

   *p = 9;
   cout << “\nThe value of n is: ” << n;
}
```

```
void dereferencingPointers()
{
   int n = 7;
   int* p = &n;

   cout << “The address of n is: ” << p;
   cout << “\nThe value of n is: ” << n;
   cout << “\nThe value of n is: ” << *p;

   n = 8;
   cout << “\nThe value of n is: ” << *p;

   *p = 9;
   cout << “\nThe value of n is: ” << n;
}
```

```
void dereferencingPointers()
{
   int n = 7;
   int* p = &n;

   cout << “The address of n is: ” << p;
   cout << “\nThe value of n is: ” << n;
   cout << “\nThe value of n is: ” << *p;

   n = 8;
   cout << “\nThe value of n is: ” << *p;

   *p = 9;
   cout << “\nThe value of n is: ” << n;
}
```

```
void dereferencingPointers()
{
   int n = 7;
   int* p = &n;

   cout << “The address of n is: ” << p;
   cout << “\nThe value of n is: ” << n;
   cout << “\nThe value of n is: ” << *p;

   n = 8;
   cout << “\nThe value of n is: ” << *p;

   *p = 9;
   cout << “\nThe value of n is: ” << n;
}
```

```
void dereferencingPointers()
{
   int n = 7;
   int* p = &n;

   cout << “The address of n is: ” << p;
   cout << “\nThe value of n is: ” << n;
   cout << “\nThe value of n is: ” << *p;

   n = 8;
   cout << “\nThe value of n is: ” << *p;

   *p = 9;
   cout << “\nThe value of n is: ” << n;
}
```

```
void dereferencingPointers()
{
   int n = 7;
   int* p = &n;

   cout << “The address of n is: ” << p;
   cout << “\nThe value of n is: ” << n;
   cout << “\nThe value of n is: ” << *p;

   n = 8;
   cout << “\nThe value of n is: ” << *p;

   *p = 9;
   cout << “\nThe value of n is: ” << n;
}
```

```
void dereferencingPointers()
{
   int n = 7;
   int* p = &n;

   cout << “The address of n is: ” << p;
   cout << “\nThe value of n is: ” << n;
   cout << “\nThe value of n is: ” << *p;

   n = 8;
   cout << “\nThe value of n is: ” << *p;

   *p = 9;
   cout << “\nThe value of n is: ” << n;
}
```

```
void dereferencingPointers()
{
   int n = 7;
   int* p = &n;

   cout << “The address of n is: ” << p;
   cout << “\nThe value of n is: ” << n;
   cout << “\nThe value of n is: ” << *p;

   n = 8;
   cout << “\nThe value of n is: ” << *p;

   *p = 9;
   cout << “\nThe value of n is: ” << n;
}
```

```
void dereferencingPointers()
{
   int n = 7;
   int* p = &n;

   cout << “The address of n is: ” << p;
   cout << “\nThe value of n is: ” << n;
   cout << “\nThe value of n is: ” << *p;

   n = 8;
   cout << “\nThe value of n is: ” << *p;

   *p = 9;
   cout << “\nThe value of n is: ” << n;
}
```

```
void dereferencingPointers()
{
   int n = 7;
   int* p = &n;

   cout << “The address of n is: ” << p;
   cout << “\nThe value of n is: ” << n;
   cout << “\nThe value of n is: ” << *p;

   n = 8;
   cout << “\nThe value of n is: ” << *p;

   *p = 9;
   cout << “\nThe value of n is: ” << n;
}
```

ARRAYS

Arrays are just pointers that point to the first element of the array.

```
void arraysArePointers()
{
   int A[3] = {1, 2, 3};

   int* p;
   p = A; //Point p at the beginning of A.

   cout << “The first element of A is located at:” << p;
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[0]); //Does the same thing as p = A.
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[1]); //Point p to the second element in A.
   cout << “\nThe second element of A is: ” << *p;
}
```

```
void arraysArePointers()
{
   int A[3] = {1, 2, 3};

   int* p;
   p = A; //Point p at the beginning of A.

   cout << “The first element of A is located at:” << p;
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[0]); //Does the same thing as p = A.
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[1]); //Point p to the second element in A.
   cout << “\nThe second element of A is: ” << *p;
}
```

```
void arraysArePointers()
{
   int A[3] = {1, 2, 3};

   int* p;
   p = A; //Point p at the beginning of A.

   cout << “The first element of A is located at:” << p;
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[0]); //Does the same thing as p = A.
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[1]); //Point p to the second element in A.
   cout << “\nThe second element of A is: ” << *p;
}
```

```
void arraysArePointers()
{
   int A[3] = {1, 2, 3};

   int* p;
   p = A; //Point p at the beginning of A.

   cout << “The first element of A is located at:” << p;
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[0]); //Does the same thing as p = A.
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[1]); //Point p to the second element in A.
   cout << “\nThe second element of A is: ” << *p;
}
```

```
void arraysArePointers()
{
   int A[3] = {1, 2, 3};

   int* p;
   p = A; //Point p at the beginning of A.

   cout << “The first element of A is located at:” << p;
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[0]); //Does the same thing as p = A.
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[1]); //Point p to the second element in A.
   cout << “\nThe second element of A is: ” << *p;
}
```

```
void arraysArePointers()
{
   int A[3] = {1, 2, 3};

   int* p;
   p = A; //Point p at the beginning of A.

   cout << “The first element of A is located at:” << p;
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[0]); //Does the same thing as p = A.
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[1]); //Point p to the second element in A.
   cout << “\nThe second element of A is: ” << *p;
}
```

```
void arraysArePointers()
{
   int A[3] = {1, 2, 3};

   int* p;
   p = A; //Point p at the beginning of A.

   cout << “The first element of A is located at:” << p;
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[0]); //Does the same thing as p = A.
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[1]); //Point p to the second element in A.
   cout << “\nThe second element of A is: ” << *p;
}
```

```
void arraysArePointers()
{
   int A[3] = {1, 2, 3};

   int* p;
   p = A; //Point p at the beginning of A.

   cout << “The first element of A is located at:” << p;
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[0]); //Does the same thing as p = A.
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[1]); //Point p to the second element in A.
   cout << “\nThe second element of A is: ” << *p;
}
```

```
void arraysArePointers()
{
   int A[3] = {1, 2, 3};

   int* p;
   p = A; //Point p at the beginning of A.

   cout << “The first element of A is located at:” << p;
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[0]); //Does the same thing as p = A.
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[1]); //Point p to the second element in A.
   cout << “\nThe second element of A is: ” << *p;
}
```

```
void arraysArePointers()
{
   int A[3] = {1, 2, 3};

   int* p;
   p = A; //Point p at the beginning of A.

   cout << “The first element of A is located at:” << p;
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[0]); //Does the same thing as p = A.
   cout << “\nThe first element of A is: ” << *p;

   p = &(A[1]); //Point p to the second element in A.
   cout << “\nThe second element of A is: ” << *p;
}
```