Addresses and Pointers

- Every byte in the computer’s memory has an *address*.
- program, when it is loaded into memory, occupies a certain range of these addresses.
- That means that every variable and every function in your program starts at a particular address.

The Address-of Operator &

- You can find the address occupied by a variable by using the *address-of* operator &.

```
// varaddr.cpp
// addresses of variables
#include <iostream>
using namespace std;
int main()
{
int var1 = 11; //define and initialize
int var2 = 22; //three variables
int var3 = 33;
```

```
 cout << &var1 << endl //print the addresses
 << &var2 << endl //of these variables
<< &var3 << endl;
 return 0;
 }
```

- **This simple program defines three integer variables and initializes them to the values 11, 22, and 33.**
- **It then prints out the addresses of these variables.**

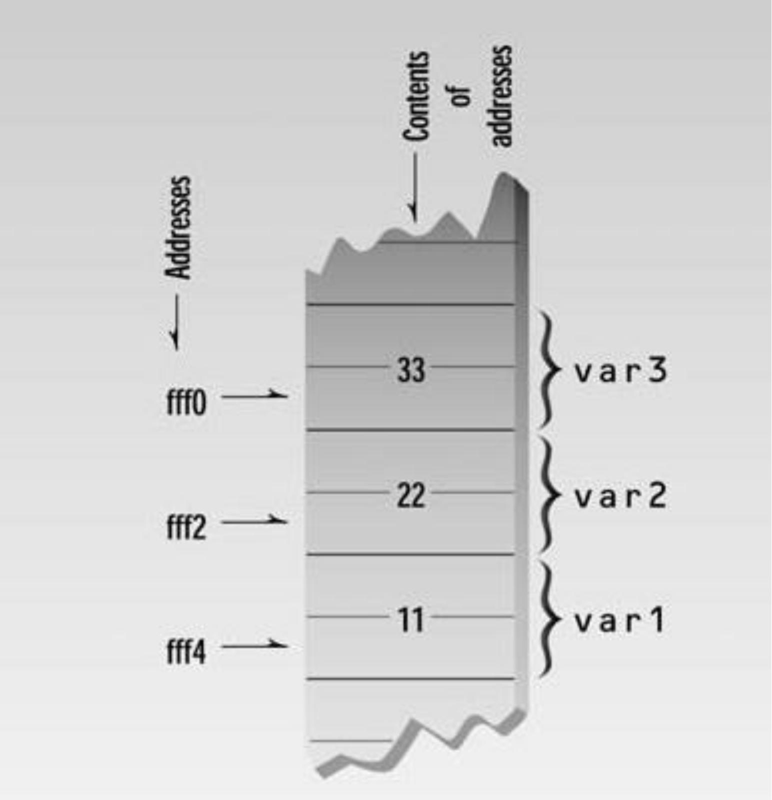
Here’s the output on our machine:

0x8f4ffff4 ← address of var1

0x8f4ffff2 ← address of var2

0x8f4ffff0 ← address of var3

- The << insertion operator interprets the addresses in hexadecimal arithmetic, as indicated by the prefix 0x before each number. This is the usual way to show memory addresses.


Addresses
Contents of addresses
fff0
fff2
fff4
33
22
11
var3
var2
var1

- The addresses appear in descending order because local variables are stored on the stack, which grows downward in memory.
- If we had used global variables, they would have ascending addresses, since global variables are stored on the heap,

Pointer Variables

- A variable that holds an address value is called a ***pointer variable***, or simply a ***pointer***.
- // ptrvar.cpp
- // pointers (address variables)
- #include <iostream>
- using namespace std;
- int main()
- {
- int var1 = 11; //two integer variables
- int var2 = 22;

```
cout << &var1 << endl //print addresses of variables
<< &var2 << endl << endl;
int* ptr;                        //pointer to integers
ptr = &var1;                     //pointer points to var1
cout << ptr << endl;      //print pointer value
ptr = &var2;                   //pointer points to var2
cout << ptr << endl;    //print pointer value
return 0;
}
```

- The program defines a ***pointer variable*** in the line
- **int*** ptr;

- The **asterisk** means ***pointer to***. Thus the statement defines the variable **ptr** as a ***pointer to*** **int**.

- char* cptr; // pointer to char
- int* iptr; // pointer to int
- float* fptr; // pointer to float
- Distance* distptr; // pointer to user-defined class Distance

- Here’s
- the output of PTRVAR:
- 0x8f51fff4 ← address of var1
- 0x8f51fff2 ← address of var2
- 0x8f51fff4 ← ptr set to address of var1
- 0x8f51fff2 ← ptr set to address of var2

Accessing the Variable Pointed To

- Suppose that we don’t know the name of a variable but we do know its address.
- There is a special syntax to access the value of a variable using its address instead of its name.
- Here’s an example program, PTRACC, that shows how it’s done:

Accessing the Variable Pointed To

- // ptracc.cpp
- // accessing the variable pointed to
- #include <iostream>
- using namespace std;
- int main()
- {
- int var1 = 11; //two integer variables
- int var2 = 22;

- int* ptr; //pointer to integers
- ptr = &var1; //pointer points to var1
- cout << *ptr << endl; //print contents of pointer (11)
- ptr = &var2; //pointer points to var2
- cout << *ptr << endl; //print contents of pointer (22)
- return 0;
- }

- This program is very similar to PTRVAR, except that instead of printing the address values in ptr, we print the integer value stored at the address that's stored in ptr. Here's the output:
- 11
- 22

- **When an asterisk is used in front of a variable name, as it is in the *ptr expression, it is called the *dereference operator* (or sometimes the *indirection operator*). It means *the value of the variable pointed to by*.**

a pointer to assign a value to a variable,

- #include <iostream>
- using namespace std;
- int main()
- {
- int var1, var2; //two integer variables
- int* ptr; //pointer to integers
- ptr = &var1; //set pointer to address of var1
- *ptr = 37; //same as var1=37
- var2 = *ptr; //same as var2=var1
- cout << var2 << endl; //verify var2 is 37
- return 0;
- }

summary

- Here’s a capsule summary of what we’ve learned so far:
- int v; //defines variable v of type int
- int* p; //defines p as a pointer to int
- p = &v; //assigns address of variable v to pointer p
- v = 3; //assigns 3 to v
- *p = 3; //also assigns 3 to v

Pointers and Arrays

- There is a close association between pointers and arrays.
- We saw how array elements are accessed.
- The following program, ARRNOTE, provides a review.

Pointers and Arrays

```
// arrnote.cpp
// array accessed with array notation
#include <iostream>
using namespace std;
int main()
{ //array
int intarray[5] = { 31, 54, 77, 52, 93 };
for(int j=0; j<5; j++) //for each element,
cout << intarray[j] << endl; //print value
return 0;
}
```

array elements can be accessed using pointer notation

```
// ptrnote.cpp
// array accessed with pointer notation
#include <iostream>
using namespace std;
int main()
{ //array
int intarray[5] = { 31, 54, 77, 52, 93 };
for(int j=0; j<5; j++) //for each element,
cout << *(intarray+j) << endl; //print value
return 0;
}
```

- The expression *(intarray+j) in PTRNOTE has exactly the same effect as intarray[j] in ARRNOTE, and the output of the programs is identical
- In next example Here we define a pointer to int—ptrint—and give it the value intarray, the address of the array.
- Now we can access the contents of the array elements with the expression*(ptrint++)

```
// ptrinc.cpp
// array accessed with pointer
#include <iostream>
using namespace std;
int main()
{
int intarray[] = { 31, 54, 77, 52, 93 }; //array
int* ptrint; //pointer to int
ptrint = intarray; //points to intarray
for(int j=0; j<5; j++) //for each element,
cout << *(ptrint++) << endl; //print value
return 0;
}
```

- The expression *(ptrint++) then represents the contents of the second array element, or 54.
- The loop causes the expression to access each array element in turn. The output of PTRINC is the same as that for PTRNOTE