**Declaring pointers.**

Due to the ability of a pointer to directly refer to the value that it points to, a pointer has different properties when it points to a *char* than when it points to an *int* or a *float*. Once dereferenced, the type needs to be known. And for that, the declaration of a pointer needs to include the data type the pointer is going to point to.

The declaration of pointers follows this syntax:

type \* name;

where type is the data type pointed to by the pointer. This type is not the type of the pointer itself, but the type of the data the pointer points to. For example:

|  |  |  |
| --- | --- | --- |
| 1 2 3 | int \* number;  char \* character;  double \* decimals; |  |

These are three declarations of pointers. Each one is intended to point to a different data type, but, in fact, all of them are pointers and all of them are likely going to occupy the same amount of space in memory (the size in memory of a pointer depends on the platform where the program runs). Nevertheless, the data to which they point to do not occupy the same amount of space nor are of the same type: the first one points to an *int*, the second one to a *char*, and the last one to a *double*. Therefore, although these three example variables are all of them pointers, they actually have different types: int\*, char\*, and double\* respectively, depending on the type they point to.

Note that the asterisk (\*) used when declaring a pointer only means that it is a pointer (it is part of its type compound specifier), and should not be confused with the *dereference operator* seen a bit earlier, but which is also written with an asterisk (\*). They are simply two different things represented with the same sign.

Let's see an example on pointers:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 | // my first pointer  #include <iostream>  using namespace std;  int main ()  {  int firstvalue, secondvalue;  int \* mypointer;  mypointer = &firstvalue;  \*mypointer = 10;  mypointer = &secondvalue;  \*mypointer = 20;  cout << "firstvalue is " << firstvalue << '\n';  cout << "secondvalue is " << secondvalue << '\n';  return 0;  } | firstvalue is 10  secondvalue is 20 | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/pointers/) |

Notice that even though neither *firstvalue* nor *secondvalue* are directly set any value in the program, both end up with a value set indirectly through the use of *mypointer*. This is how it happens: First, *mypointer* is assigned the address of *firstvalue* using the address-of operator (&). Then, the value pointed to by *mypointer* is assigned a value of 10. Because, at this moment, *mypointer* is pointing to the memory location of *firstvalue*, this in fact modifies the value of *firstvalue*.

In order to demonstrate that a pointer may point to different variables during its lifetime in a program, the example repeats the process with *secondvalue* and that same pointer, *mypointer*.

Here is an example a little bit more elaborated:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 | // more pointers  #include <iostream>  using namespace std;  int main ()  {  int firstvalue = 5, secondvalue = 15;  int \* p1, \* p2;  p1 = &firstvalue; // p1 = address of firstvalue  p2 = &secondvalue; // p2 = address of secondvalue  \*p1 = 10; // value pointed to by p1 = 10  \*p2 = \*p1; // value pointed to by p2 = value pointed to by p1  p1 = p2; // p1 = p2 (value of pointer is copied)  \*p1 = 20; // value pointed to by p1 = 20    cout << "firstvalue is " << firstvalue << '\n';  cout << "secondvalue is " << secondvalue << '\n';  return 0;  } | firstvalue is 10  secondvalue is 20 | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/pointers/) |

Each assignment operation includes a comment on how each line could be read: i.e., replacing ampersands (&) by "address of", and asterisks (\*) by "value pointed to by".

Notice that there are expressions with pointers p1 and p2, both with and without the *dereference operator* (\*). The meaning of an expression using the *dereference operator* (\*) is very different from one that does not. When this operator precedes the pointer name, the expression refers to the value being pointed, while when a pointer name appears without this operator, it refers to the value of the pointer itself (i.e., the address of what the pointer is pointing to).

Another thing that may call your attention is the line:

|  |  |  |
| --- | --- | --- |
|  | int \* p1, \* p2; |  |

This declares the two pointers used in the previous example. But notice that there is an asterisk (\*) for each pointer, in order for both to have type int\* (pointer to int). This is required due to the precedence rules. Note that if, instead, the code was:

|  |  |  |
| --- | --- | --- |
|  | int \* p1, p2; |  |

p1 would indeed be of type int\*, but p2 would be of type int. Spaces do not matter at all for this purpose. But anyway, simply remembering to put one asterisk per pointer is enough for most pointer users interested in declaring multiple pointers per statement. Or even better: use a different statement for each variable.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | // references and pointers  #include <cstdio>  int main(int argc, char \*\*argv)  {  int x = 5;  int \*ip = &x; // pointer connection  int &y = x; // reference connection  printf("The value of x is %d\n", x); // 5  printf("The value of \*ip is %d\n", \*ip); // 5  printf("The value of y is %d\n", y); // 5  printf("\n");  x = 64;  printf("The value of x is %d\n", x); // 64  printf("The value of \*ip is %d\n", \*ip); // 64  printf("The value of y is %d\n", y); // 64  printf("\n");  y = 27;  printf("The value of x is %d\n", x); // 27  printf("The value of \*ip is %d\n", \*ip); // 27  printf("The value of y is %d\n", y); // 27  printf("\n");  \*ip = 33;  printf("The value of x is %d\n", x); // 33  printf("The value of \*ip is %d\n", \*ip); // 33  printf("The value of y is %d\n", y); // 33  printf("\n");  int z = 123; ip = &z;  printf("The value of x is %d\n", x); // 33  printf("The value of \*ip is %d\n", \*ip); // 123  printf("The value of y is %d\n", y); // 33  printf("The value of z is %d\n", z); // 123  printf("\n");  y = z;  printf("The value of x is %d\n", x); // 123  printf("The value of \*ip is %d\n", \*ip); // 123  printf("The value of y is %d\n", y); // 123  printf("The value of z is %d\n", z); // 123  printf("\n");  return 0;  }  /\*  The main differences between pointers and references are:  1) References are used to refer an existing variable in another name  whereas pointers are used to store address of variable.  2) References cannot have a null value assigned but pointer can.  3) A reference variable can be referenced by pass by value  whereas a pointer can be referenced but pass by reference.  5) A reference must be initialized on declaration  while it is not necessary in case of pointer.  6) A reference shares the same memory address with the original variable  but also takes up some space on the stack  whereas a pointer has its own memory address and size on the stack.  \*/ |

\*Check below written code in Standard C:

Pointers:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 | #include "iostream"  using namespace std;  int main()  {  int \*ptr;  cout << \*ptr << endl;  return 0;  } | Exception has occurred: W32/0xC0000005  Unhandled exception thrown: read access violation. ptr was nullptr.  Uninitialized local variable ‘ptr’ used. | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/ntcs/) |
| 1 2 3 4 5 6 7 8 | #include "iostream"  using namespace std;  int main()  {  int \*ptr;  return 0;  } | Press any key to continue . . . | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/ntcs/) |
| 1 2 3 4 5 6 7 8 | #include "iostream"  using namespace std;  int main()  {  int \*ptr;  cout << ptr << endl;  return 0;  } | 00000000  Uninitialized local variable ‘ptr’ used. | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/ntcs/) |

References:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 | #include "iostream"  using namespace std;  int main()  {  int &ref;  return 0;  } | ‘ref’:References must be initialized.  Reference variable “ref” requires an initializer. | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/ntcs/) |
| 1 2 3 4 5 6 7 8 | #include "iostream"  using namespace std;  int main()  {  int var;  int &ref = var;  return 0;  } | Press any key to continue . . . | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/ntcs/) |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | #include "iostream"  using namespace std;  int main()  {  int \*ptr;  int var = 1;  int foo = 8;  ptr = &var; // pointer to the address of var  // now var is the same as ptr  cout << var << endl;  cout << &var << endl;  cout << \*ptr << endl;  cout << ptr << endl << endl;  ptr = &foo; // pointer to the address of foo  // now foo is the same as ptr  cout << foo << endl;  cout << &foo << endl;  cout << \*ptr << endl;  cout << ptr << endl << endl;  int &ref = var; //reference to var, now var is the same as ref  cout << var << endl;  cout << &var << endl;  cout << ref << endl;  cout << &ref << endl << endl;  ref = 10;  cout << var << endl;  cout << &var << endl;  cout << ref << endl;  cout << &ref << endl << endl;  var = 20;  cout << var << endl;  cout << &var << endl;  cout << ref << endl;  cout << &ref << endl << endl;  // int &ref = foo;  // cout << foo << endl;  // cout << &foo << endl;  // cout << ref << endl;  // cout << &ref << endl << endl;  return 0;  } | 1  005AFAC8  1  005AFAC8  8  005AFABC  8  005AFABC  1  005AFAC8  1  005AFAC8  10  005AFAC8  10  005AFAC8  20  005AFAC8  20  005AFAC8 | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/ntcs/) |

The main difference is deduced in the fact where we can reuse, repoint pointer to another address, however we can make reference when declaring and initializing (declaring and initializing at the same time) it with variable only once. One reference per variable. Which means if uncomment codes in lines 35 to 39 inclusively, our program won’t run because compiler after encountering 35th line will pop out error about ‘ref’:redefinition; multiple initialization, or constant, non-modifiable value should assigned without redefinition.

Below two the same codes describing pointers. Yet pointer *\*ptr* is initialized and declared differently. On the left it is declared and initialized with *&foo* at the same time simultaneously. However, on the right-side code the *\*ptr* first declared and the initialized to *&foo* on the next line of code. Again, remember we are not allowed to such stuff with references.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | #include "iostream"  using namespace std;  int main()  {  int foo = 8;  int \*ptr = &foo;  cout << foo << endl;  cout << &foo << endl;  cout << \*ptr << endl;  cout << ptr << endl;  cout << &ptr << endl; \*ptr = 1;  cout << foo << endl;  cout << &foo << endl;  cout << \*ptr << endl;  cout << ptr << endl;  cout << &ptr << endl;  foo = 20;  cout << foo << endl;  cout << &foo << endl;  cout << \*ptr << endl;  cout << ptr << endl;  cout << &ptr << endl;  return 0;  } | #include "iostream"  using namespace std;  int main()  {  int foo = 8;  int \*ptr;  ptr = &foo;  cout << foo << endl;  cout << &foo << endl;  cout << \*ptr << endl;  cout << ptr << endl;  cout << &ptr << endl;  \*ptr = 1;  cout << foo << endl;  cout << &foo << endl;  cout << \*ptr << endl;  cout << ptr << endl;  cout << &ptr << endl;  foo = 20;  cout << foo << endl;  cout << &foo << endl;  cout << \*ptr << endl;  cout << ptr << endl;  cout << &ptr << endl;  return 0;  } |  |
|  | Output:  8  008FFE98  8  008FFE98  008FFE9C  1  008FFE98  1  008FFE98  008FFE9C  20  008FFE98  20  008FFE98  008FFE9C | Output:  8  00F8FD68  8  00F8FD68  00F8FD6C  1  00F8FD68  1  00F8FD68  00F8FD6C  20  00F8FD68  20  00F8FD68  00F8FD6C |  |

One more thing to practice is that ‘*int\* ptr’* and ‘*int& ref’* are both type of “int \*”. However below code won’t run:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 | #include <iostream>  using namespace std;  int main ()  {  int var = 8;  int \* ptr;  \*ptr = &var;  } | '=': cannot convert from 'int \*' to 'int'  a value of type "int \*" cannot be assigned to an entity of type "int" | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/pointers/) |
| 1 2 3 4 5 6 7 8 | #include <iostream>  using namespace std;  int main ()  {  int var = 8;  int \* ptr;  \*ptr = var;  } | Exception has occurred: W32/0xC0000005  Unhandled exception thrown: write access violation.  ptr was 0xE31113.  uninitialized local variable 'ptr' used | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/pointers/) |
| 1 2 3 4 5 6 7 8 | #include <iostream>  using namespace std;  int main ()  {  int var = 8;  int \* ptr;  ptr = var;  } | '=': cannot convert from 'int \*' to 'int'  a value of type "int \*" cannot be assigned to an entity of type "int" | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/pointers/) |

Below however, two codes run and debug, and actually are the same:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 | #include <iostream>  using namespace std;  int main ()  {  int var = 8;  int \*ptr;  ptr = &var;  } | Press any key to continue . . . | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/pointers/) |
| 1 2 3 4 5 6 7 8 | #include <iostream>  using namespace std;  int main ()  {  int var = 8;  int \*ptr = &var;  } | Press any key to continue . . . | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/pointers/) |