Pointers and References

LAB 13: Polynomial Showdown using a Linked-List

Rung-Bin Lin

International Bachelor Program in Informatics
Yuan Ze University

12/19/2023

Pointers

- Pointer?
  - **A variable used to store the address of other variable. Hence, a pointer variable also has its own address.**
  - **A pointer should be initialized either when they are declared or in an assignment.**

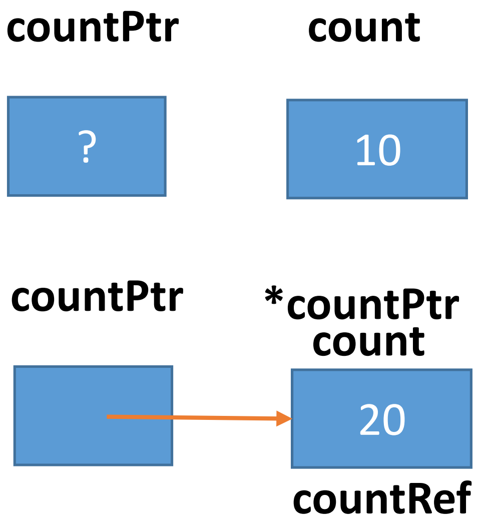
```
int  count = 10, *countPtr, *aryPtr, *aryElt;
```

- Reference?

Address-of operator

```
int *countPtr=&count; // initializing countPtr
int &countRef = count;
*countPtr = 20;
cout << count << *countPtr; // what are printed?
```

Dereferencing a pointer



- Array address

```
int intAry[20];  // array name is a constant  pointer
int *aryPtr = intAry;
int *aryElt = &intAry[12];
```

Four Types of Pointers

- Nonconstant pointer to nonconstant data
  - Pointer and data both can be modified

    ```
    int *countPtr;
    ```

- Nonconstant pointer to constant data
  - Pointer can be modified but data cannot

    ```
    const int *countPtr; // may not initialize pointer on declaration.
    ```

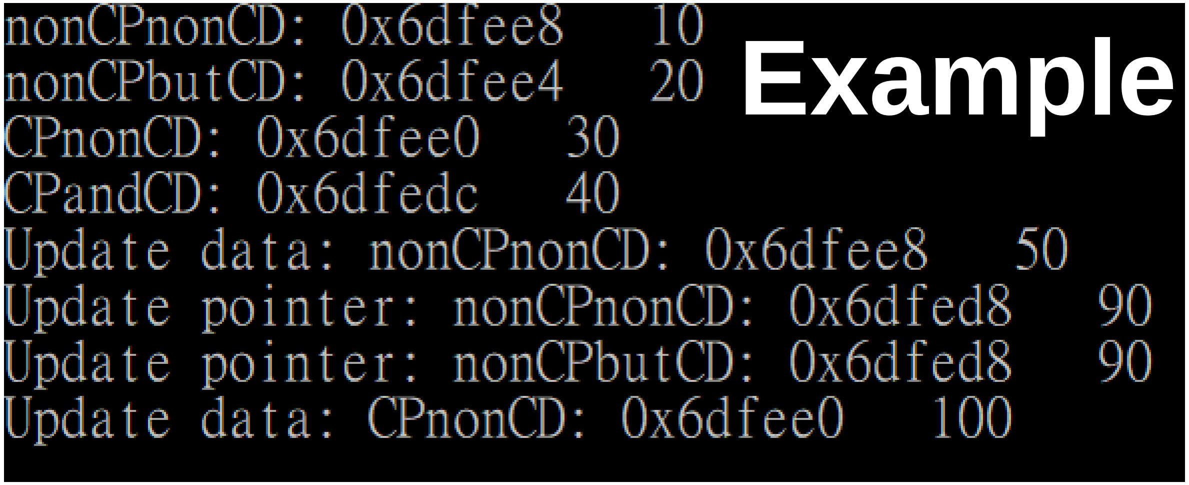
- Constant pointer to nonconstant data
  - Data can be modified but pointer cannot

    ```
    int x;
    int * const countPtr = &x;  // Pointer must be initialized on declaration.
    ```

- Constant pointer to constant data
  - Pointer and data cannot be modified

    ```
    const int x=5;
    const int *const countPtr = &x;  // must be initialized
    ```

Example



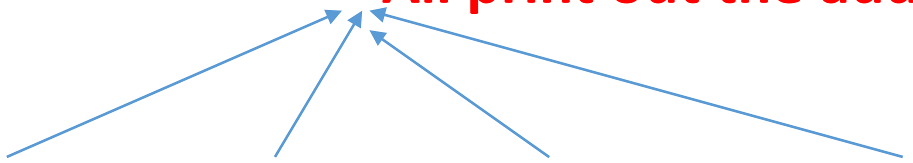
```
nonCPnonCD: 0x6dfee8   10
nonCPbutCD: 0x6dfee4   20
CPnonCD: 0x6dfee0   30
CPandCD: 0x6dfedc   40
Update data: nonCPnonCD: 0x6dfee8   50
Update pointer: nonCPnonCD: 0x6dfed8   90
Update pointer: nonCPbutCD: 0x6dfed8   90
Update data: CPnonCD: 0x6dfee0   100
```

```
int main(){
   int anInt = 10;
   int xInt = 20;
   int yInt = 30;
   int zInt = 40;
   int *nonCPnonCD = &anInt;
   const int *nonCPbutCD = &xInt;
   int *const CPnonCD = &yInt;
   const int * const CPandCD = &zInt;
   cout << "nonCPnonCD: " << nonCPnonCD << "   " <<  *nonCPnonCD << endl;
   cout << "nonCPbutCD: " <<nonCPbutCD << "   " <<  *nonCPbutCD << endl;
   cout << "CPnonCD: " << CPnonCD << "   " <<  *CPnonCD << endl;
   cout << "CPandCD: " << CPandCD << "   " <<  *CPandCD << endl;
   *nonCPnonCD = 50;
   cout << "Update data: " << "nonCPnonCD: " << nonCPnonCD << "   " <<  *nonCPnonCD << endl;
   int yy;
   int *tempD = &yy;
   *tempD = 90;
   nonCPnonCD = tempD;
   cout << "Update pointer: " << "nonCPnonCD: " << nonCPnonCD << "   " <<  *nonCPnonCD << endl;
   //*nonCPbutCD = 90;  // cannot modify data
   nonCPbutCD = &yy;
   cout << "Update pointer: " << "nonCPbutCD: " << nonCPbutCD << "   " <<  *nonCPbutCD << endl;
   //CPnonCD = &tempT; // cannot modify pointer
   *CPnonCD = 100;
   cout << "Update data: " << "CPnonCD: " << CPnonCD << "   " <<  *CPnonCD << endl;
   //*CPandCD = 100;    //  cannot modify data
   //CPandCD = &yy;    // cannot modify pointer
}
```

Pointers & References

```
int a;
int *aPtr;  // aPtr is a variable whose value is
            // an address that can hold an integer.
a=7;
aPtr = &a;
```

All print out the address of a.

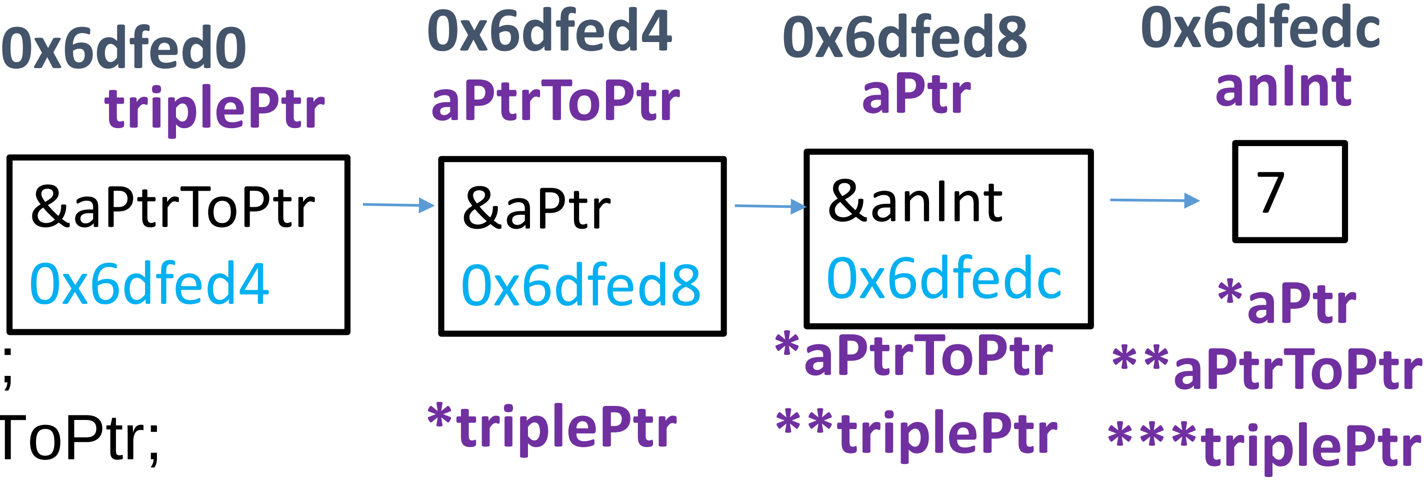


```
cout << &a << aPtr << *&aPtr << &*aPtr;
cout << a << *aPtr ;
```

The value stored in the address (location) pointed by aPtr.

Pointer Pointing to a Pointer

```
int anInt=7;
int *aPtr = &anInt;
int **aPtrToPtr =&aPtr;
int ***triplePtr = &aPtrToPtr;
```



0x6dfed0
triplePtr
&aPtrToPtr
0x6dfed4

0x6dfed4
aPtrToPtr
&aPtr
0x6dfed8
*triplePtr

0x6dfed8
aPtr
&anInt
0x6dfedc
*aPtrToPtr
**triplePtr

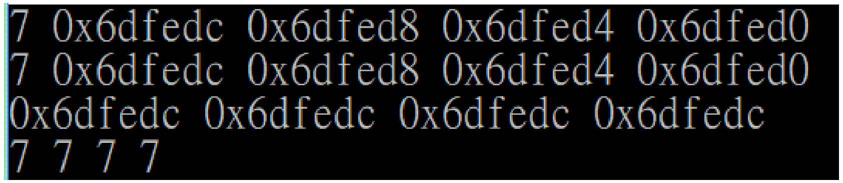
0x6dfedc
anInt
7
*aPtr
**aPtrToPtr
***triplePtr

aPtrToPtr is a pointer that points to the pointer aPtr. aPtr is a pointer that points to the integer anInt.

That is to say, aPtrToPtr is an address where stores the address of aPtr, and aPtr is an address where stores the address of anInt. Try below.

```
cout << ***triplePtr << " " << **triplePtr << " " << *triplePtr << " "  << triplePtr << " " << &triplePtr <<  endl;
cout << anInt << " " << aPtr << " " << aPtrToPtr << " " << triplePtr << " " << &triplePtr << endl;
cout << &anInt << " " << aPtr << " " << *aPtrToPtr << " " << **triplePtr << endl;
cout << anInt << " " << *aPtr << " " << **aPtrToPtr << " " << ***triplePtr << endl;
```

This is the output:



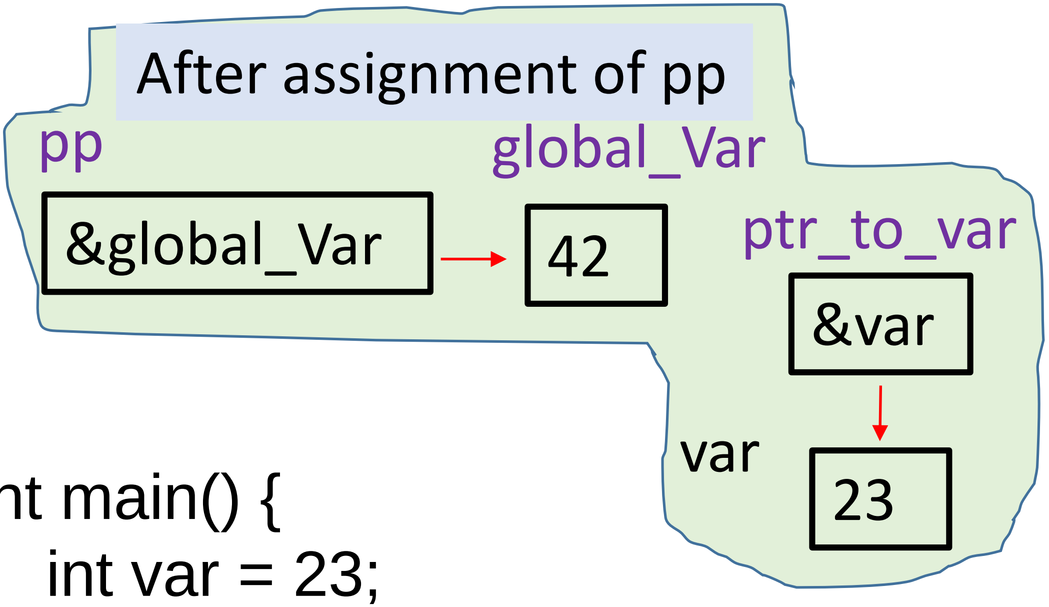
```
7 0x6dfedc 0x6dfed8 0x6dfed4 0x6dfed0
7 0x6dfedc 0x6dfed8 0x6dfed4 0x6dfed0
0x6dfedc 0x6dfedc 0x6dfedc 0x6dfedc
7 7 7 7
```

Passing Reference to a Pointer in C++

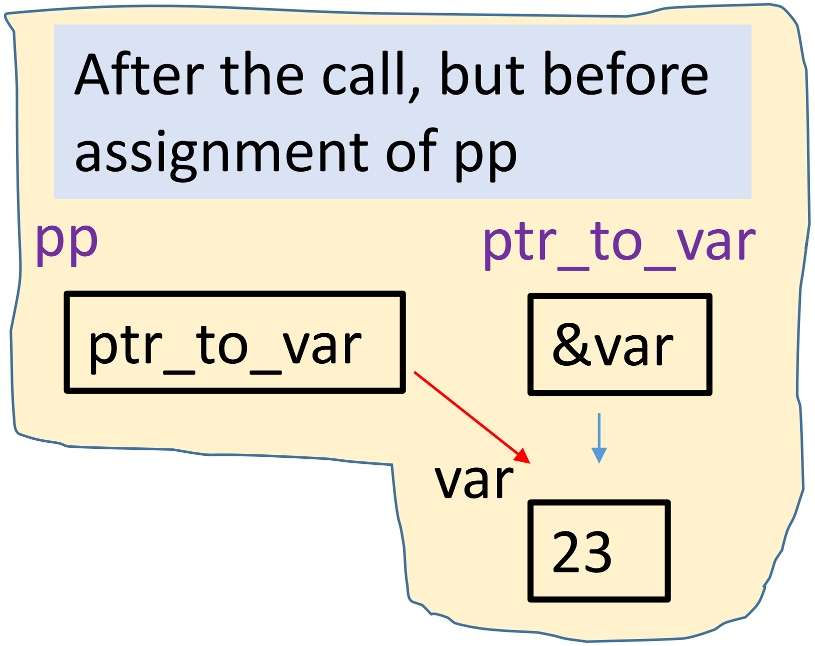
https://www.geeksforgeeks.org/passing-reference-to-a-pointer-in-c/

Example for Modifying Pointer Pointing to a Pointer in a Function: Not working

```
int global_Var = 42;
// function to change pointer value
void changePointerValue(int *pp) {
    pp = &global_Var;
}
```



After assignment of pp



After the call, but before assignment of pp

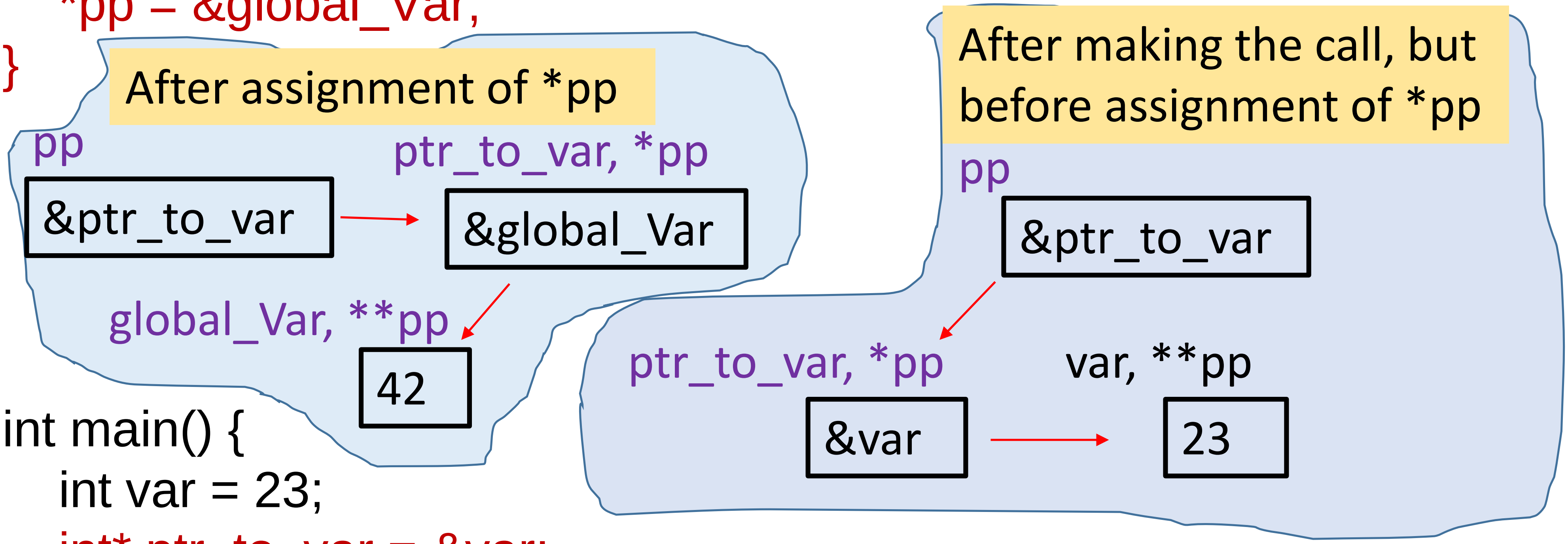
We Intend to modify the pointer stored in ptr_to_var from &var to &global_Var.

```
int main() {
    int var = 23;
    int *ptr_to_var = &var;
    cout << "Passing Pointer to function:" << endl;
    cout << "Before :" << *ptr_to_var << endl; // display 23
    changePointerValue(ptr_to_var);
    cout << "After :" << *ptr_to_var << endl; // display 23
}
```

Example for Modifying Pointer Pointing to a Pointer in a Function: Working Fine

```
int global_Var = 42;
// function to change pointer value
void changePointerValue( int** pp) {
    *pp = &global_Var;
}
```

This enables us to modify the pointer stored in ptr_to_var from &var to &global_Var.



After assignment of *pp

After making the call, but before assignment of *pp

```
int main() {
    int var = 23;
    int* ptr_to_var = &var;
    cout << "Passing Pointer to function:" << endl;
    cout << "Before :" << *ptr_to_var << endl; // display 23
    changePointerValue( &ptr_to_var );
    cout << "After :" << *ptr_to_var << endl; // display 42
}
```

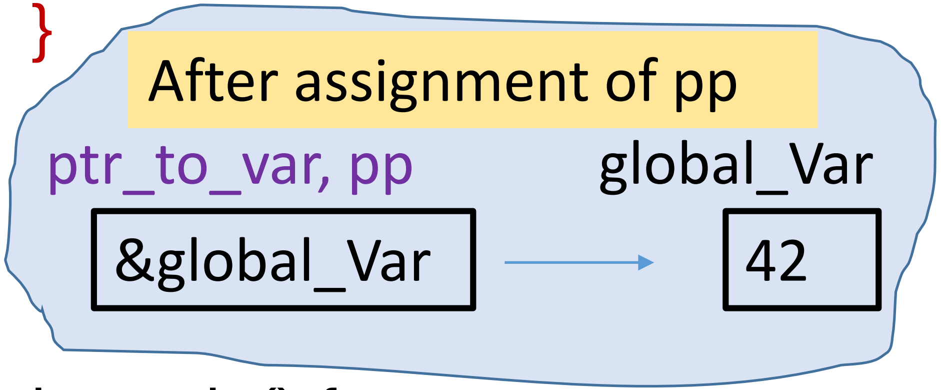
In Summary, passing the address of a pointer to the function will work.

Example for Modifying a Reference to a Pointer in a Function: Working Fine

```
int global_Var = 42;
// function to change pointer value
// pp is a reference to a pointer to an integer
void changePointerValue( int *&pp) {
pp = &global_Var;
}
```

PP is a reference to an int pointer.

This also enables us to modify the pointer stored in ptr_to_var from &var to &global_Var.



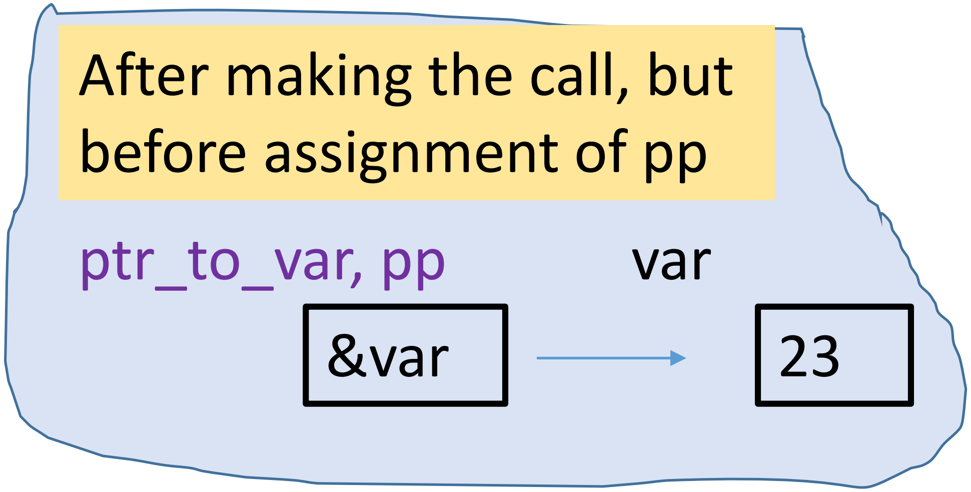
After assignment of pp

ptr_to_var, pp

global_Var

&global_Var

42



After making the call, but before assignment of pp

ptr_to_var, pp

var

&var

23

```
int main() {
    int var = 23;
    int *ptr_to_var = &var;
    cout << "Passing Pointer to function:" << endl;
    cout << "Before :" << *ptr_to_var << endl; // display 23
    changePointerValue( ptr_to_var );
    cout << "After :" << *ptr_to_var << endl; // display 42
}
```

Pass-by-reference vs. Pass-by-value

```
int cubeByValue( int );

int main()
{
   int number = 5;
   number =
cubeByValue( number );

} // end main

int cubeByValue( int n )
{
   return n * n * n;
}
```

```
void cubeByReference( int * );

int main()
{
   int number = 5;
   cubeByReference( &number );
} // end main

void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

sizeof Operator

```
char c; // variable of type char
short s; // variable of type short
int i; // variable of type int
long l; // variable of type long
float f; // variable of type float
double d; // variable of type double
long double ld; // type long double
int array[ 20 ]; // array of int
int *ptr = array; // type int *
```

sizeof is an operator that gives the number of bytes taken by a type or a variable.

```
    cout << "sizeof c = " << sizeof c
     << "\tsizeof(char) = " << sizeof( char )
     << "\nsizeof s = " << sizeof s
    << "\tsizeof(short) = " << sizeof( short )
    << "\nsizeof i = " << sizeof i
    << "\tsizeof(int) = " << sizeof( int )
    << "\nsizeof l = " << sizeof l
    << "\tsizeof(long) = " << sizeof( long )
   << "\nsizeof f = " << sizeof f
   << "\tsizeof(float) = " << sizeof( float )
  << "\nsizeof d = " << sizeof d
     << "\tsizeof(double) = " << sizeof( double )
      << "\nsizeof ld = " << sizeof ld
     << "\tsizeof(long double) = " << sizeof( long
double )
<< "\nsizeof array = " << sizeof array
<< "\nsizeof ptr = " << sizeof ptr << endl;
```

Pointer & Array

```
int main()
{
  int b[] = { 10, 20, 30, 40 };
  int *bPtr = b;

  for ( int i = 0; i < 4; i++ )
    cout << "b[" << i << "] = " << b[ i ] << '\n';

  for ( int offset1 = 0; offset1 < 4; offset1++ )
    cout << "*(b + " << offset1 << ") = " << *( b + offset1 ) << '\n';

  for ( int j = 0; j < 4; j++ )
    cout << "bPtr[" << j << "] = " << bPtr[ j ] << '\n';

  for ( int offset2 = 0; offset2 < 4; offset2++ )
    cout << "*(bPtr + " << offset2 << ") = "
      << *( bPtr + offset2 ) << '\n';

} // end main
```

This is so-called pointer arithmetic, especially when used along with an array. Hence, b+offset1 refers to the (offset1)-th element in array b.

Pointer-Based string Processing

```
char color[ ]=“blue”;
const char *colorPtr = “blue”;
char colorx[ ]= {‘b’, ‘l’, ‘u’, ‘e’, ‘\0’};
const char* const suit[4] = {“Hearts”, “Diamonds”,
“Clubs”, “Spades”};  // Array of pointers
```

Function Pointers (7.12)

```
// prototypes
void selectionSort( int [ ], const int, bool (*)( int, int ) );
void swap( int * const, int * const );
bool ascending( int, int ); // implements ascending order
bool descending( int, int ); // implements descending order

Int main (){
 if ( order == 1 )
      selectionSort( a, arraySize, ascending );
   else
      selectionSort( a, arraySize, descending );
…..
}
```

Function pointer

```
void selectionSort( int work[ ], const int size, bool (*compare)( int, int ) )
{

}
```

Elaboration

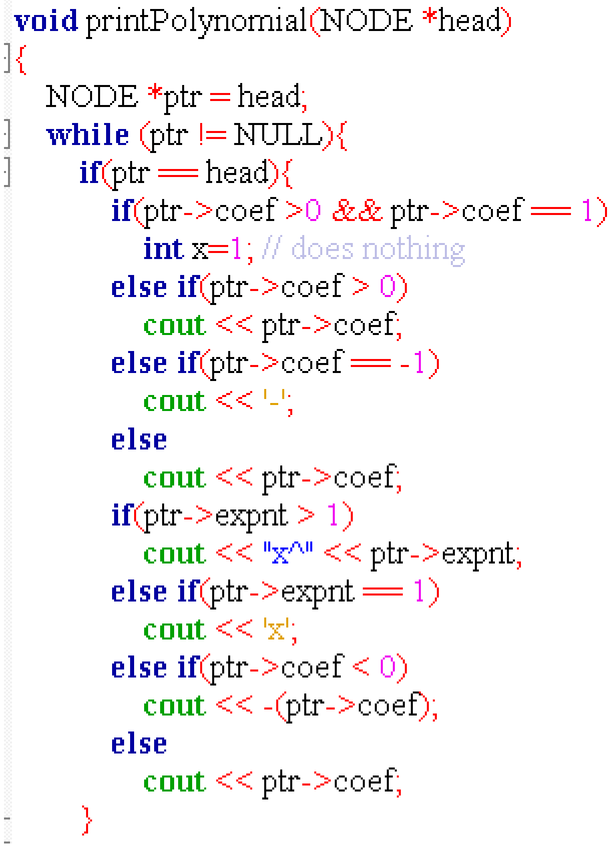
- With the ***new*** keyword...
  - MyClass* myClass = new MyClass();
  - myClass->MyField = "Hello world!";
- Without the ***new*** keyword...
  - MyClass myClass;
  - myClass.MyField = "Hello world!";
- What are the differences between with and without using ***new*** function?
  - https://stackoverflow.com/questions/655065/when-should-i-use-the-new-keyword-in-c

LAB 13: Polynomial Showdown using a Linked-List

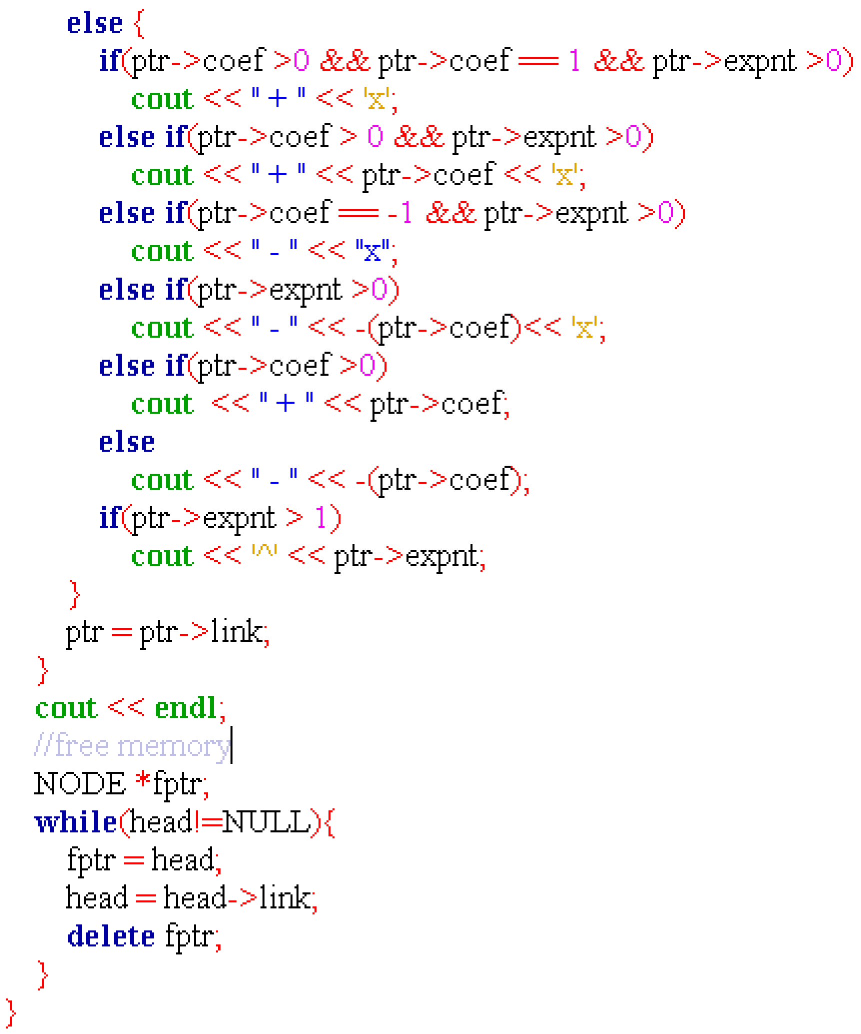
- Given the coefficients of a polynomial from degree *n* down to 0, you are asked to format the polynomial in a readable format with unnecessary characters removed. For instance, given the coefficients 0, 0, 0, 1, 22, -333, 0, 1, and -1, you should generate an output line which displays x^5 + 22x^4 - 333x^3 + x - 1. The following formatting rules must be adhered:

1. Terms must appear in decreasing order of degree.
2. Exponents should appear after a caret “^”.
3. The constant term appears as only the constant.
4. Only terms with nonzero coefficients should appear, unless all terms have zero coefficients in which case the constant term should appear.
5. The only spaces should be a single space on either side of the binary + and − operators.
6. If the leading term is positive, no sign should precede it; a negative leading term should be preceded by a minus sign, as in -7x^2 + 30x + 66.
7. Negated terms should appear as a subtracted unnegated term (with the exception of a negative leading term which should appear as described above). That is, rather than x^2 + -3x, the output should be x^2 - 3x.
8. The constants 1 and -1 should appear only as the constant term. That is, rather than -1x^3 + 1x^2 + 3x^1 - 1, the output should appear as -x^3 + x^2 + 3x - 1.

- What you need to do is to write a main() function to create a linked list that will store a polynomial read from the keyboard. After a linked list is created, you should make a call to **void printPolynomial(NODE *)** to print out a polynomial where NODE is a *struct* used to store a term in a polynomial. Below is the code of this function which should not be modified. The parameter head is the head of a linked list.



```
void printPolynomial(NODE *head)
{
  NODE *ptr = head;
  while (ptr != NULL){
    if(ptr == head){
      if(ptr->coef >0 && ptr->coef == 1)
        int x=1; // does nothing
      else if(ptr->coef > 0)
        cout << ptr->coef;
      else if(ptr->coef == -1)
        cout << '-';
      else
        cout << ptr->coef;
      if(ptr->expnt > 1)
        cout << "x^" << ptr->expnt;
      else if(ptr->expnt == 1)
        cout << 'x';
      else if(ptr->coef < 0)
        cout << -(ptr->coef);
      else
        cout << ptr->coef;
    }
```

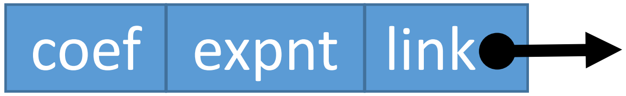


```
    else {
      if(ptr->coef >0 && ptr->coef == 1 && ptr->expnt >0)
        cout << " + " << 'x';
      else if(ptr->coef > 0 && ptr->expnt >0)
        cout << " + " << ptr->coef << 'x';
      else if(ptr->coef == -1 && ptr->expnt >0)
        cout << " - " << "x";
      else if(ptr->expnt >0)
        cout << " - " << -(ptr->coef)<< 'x';
      else if(ptr->coef >0)
        cout  << " + " << ptr->coef;
      else
        cout << " - " << -(ptr->coef);
      if(ptr->expnt > 1)
        cout << '^' << ptr->expnt;
    }
    ptr = ptr->link;
  }
  cout << endl;
  //free memory
  NODE *fptr;
  while(head!=NULL){
    fptr = head;
    head = head->link;
    delete fptr;
  }
}
```

Use of struct to Define a node

- **struct** is a language construct used to create a user-defined data type that can hold several data items of different types together as a whole. For example, we can define a type of NODE as follows:

```
struct NODE {
     int coef;
     int expnt;
     NODE *link;
}
```



coef expnt link

The above struct can be used to define a node (variable) that can be used to store a term of a polynomial where *coef* is used to store the coefficient, *expnt* is used to store the exponent of a polynomial, and *link* as the above figure shown which can be used to link to the next node.

```
NODE aNode; // aNode is a node.
NODE *nodePtr; // A pointer pointing to a node of type NODE
```

Creating a Node

- **Two ways: declaration and dynamic allocation**
- **Declaration**

```
NODE aNode; // aNode is a node.
aNode.coef = coef; // store coefficient into the node
aNode.expnt = expnt; //store exponent into the node
aNode.link = NULL; // set the link of the node to NULL
```

- **Dynamic allocation**

```
NODE *aNode;  // aNode is a pointer to a node
aNode = new NODE;  // create a node pointed by aNode
aNode->coef = coef;  // store coefficient into the node
aNode.expnt = expnt; //store exponent into the node
aNode->link = NULL; // set the link of the node to NULL
```

- If aNode is a pointer, use -> to get access to a member of a node pointed by aNode. Note that aNode must already point to an existing node. If aNode is a node, use . to get access to a member of aNode.
- **What are the differences between with and without using new function?** https://stackoverflow.com/questions/655065/when-should-i-use-the-new-keyword-in-c

Input and Output formats

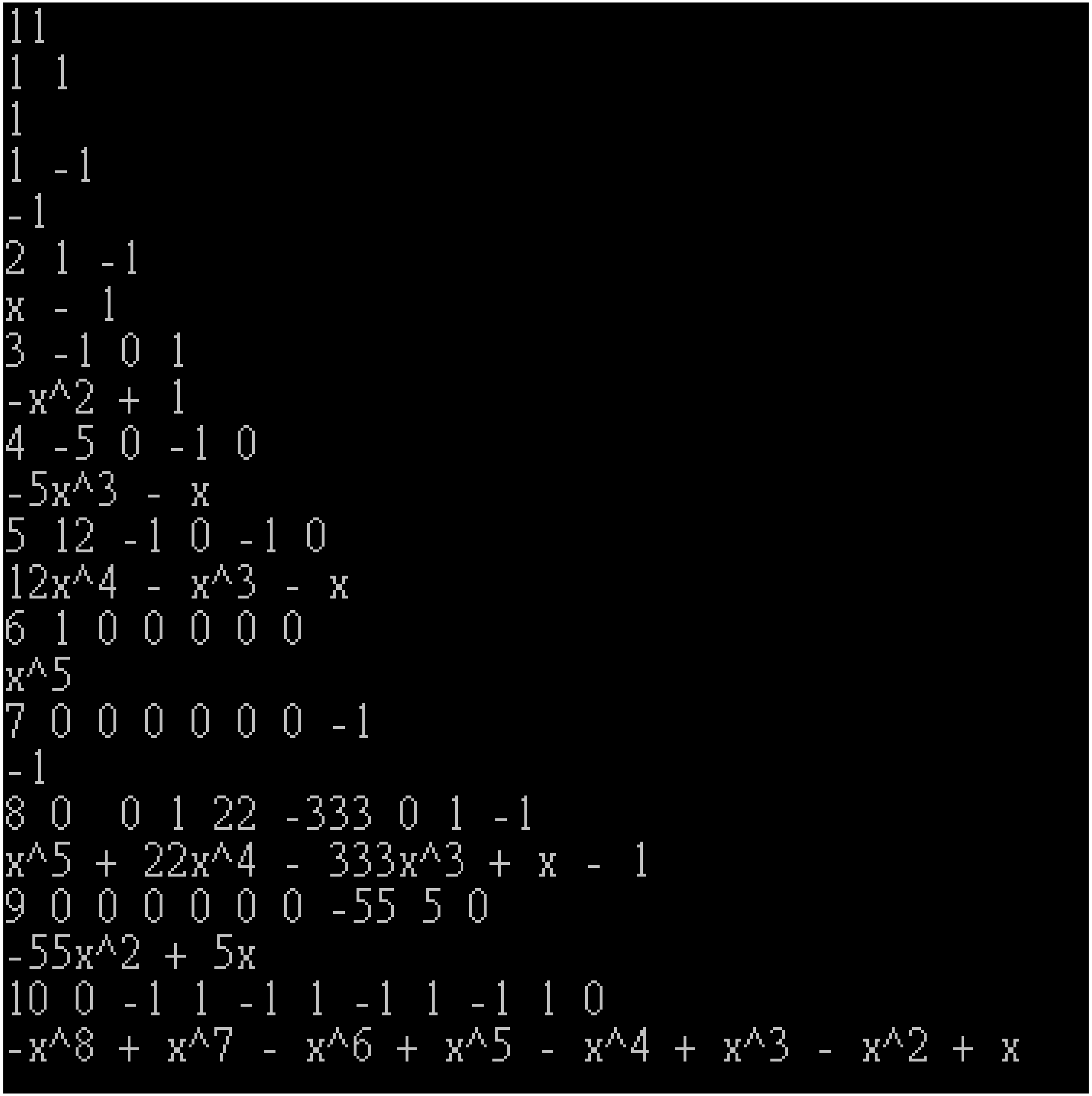
- **Input format**

  The first line gives the number of test cases. The input of each test case takes a line that holds a sequence of integers. The first number in the sequence gives the number of coefficients for the terms in the polynomial. The remaining numbers are coefficients themselves.

- **Output format**

  The output of a test case takes a line that gives the corresponding readable polynomial. The output has been handled by the given function. Hence, you need just make a call to printPolynomial(…).

Sample Input & Output



```
11
1 1
1
1 -1
-1
2 1 -1
x - 1
3 -1 0 1
-x^2 + 1
4 -5 0 -1 0
-5x^3 - x
5 12 -1 0 -1 0
12x^4 - x^3 - x
6 1 0 0 0 0 0
x^5
7 0 0 0 0 0 0 -1
-1
8 0  0 1 22 -333 0 1 -1
x^5 + 22x^4 - 333x^3 + x - 1
9 0 0 0 0 0 0 -55 5 0
-55x^2 + 5x
10 0 -1 1 -1 1 -1 1 -1 1 0
-x^8 + x^7 - x^6 + x^5 - x^4 + x^3 - x^2 + x
```

Requirements

- The main() function should make a call only to **printPolynomial(…)** to print a polynomial in readable format.
- The function **printPolynomial(…)** should not be modified.

Hints

- Should have a pointer pointing to the head of a linked list.
- Had better set the *link* field of a node to NULL when a node is initialized with data.
- Always insert a node after the **tail node** of a linked list, i.e., at the end of a linked list.
- Maintain a pointer that always points to the last node of a linked list so that inserting a node at the end of a linked list can be done easily.