PFI lecture POINTERS

- This is the first slide in the variable slides and let us review it
- The concept of variable in programming
  - A location in memory for storing information
  - It has a name
  - It has two values: the information stored and memory location
  - It has a type
    - For whole number or integer: int
    - For “real number” (its approximate called “floating point”): double
    - For charater: char
    - For string of characters (e.g. statement in English): string
    - For decision making (as in if statement or loop): bool
  - It must be declared or introduced first to be able to use it
  - Programmers decide what variables to use and their names, types
  - It has a scope and storage class (for now without these concepts we may still comprehend the c++ code), will discuss them later.

PFI lecture POINTERS

- Pointer is a memory location in which the address of another memory location is stored.
- Pointer is a variable that stores the address of another memory location.
- The value of a pointer is an address of another memory.
- We say a pointer **points to** another memory if the address of that memory is stored in the pointer.
- The **type** of a pointer depends on the type of value stored at the memory location to which the pointer points.

PFI lecture POINTERS

- How do we declare a pointer variable? Well it depends on the type of value stored at the location to which the pointer points.
- Examples:
  - Pointer to int, the location pointed by the pointer stores int

  int * ptr_to_int;
  - Pointer to char

  char * prt_to_char;
  - Pointer to TYPE, TYPE stands for any type built-in or user defined (class, struct)

  TYPE * ptr_to_TYPE;

PFI lecture POINTERS

- How do we assign values a pointer variable? Or where and how do we get the address of another memory location? Remember each variable refers to a memory location. There is an operator (&) that "retrieves" the address of a variable for us.
- Examples:
  - Pointer to int

```
int * ptr_to_int;
int n=7;
ptr_to_int = &n; // ptr_to_int points to the memory
   location that n is associated with
```

PFI lecture POINTERS

- How do we assign values a pointer variable? Or where and how do we get the address of another memory location? Remember an array name is a pointer to the array element type.
- Examples:
  - Pointer to int

  int * ptr_to_int;

  int n[7];

  ptr_to_int = n; // ptr_to_int points to the memory location that n[0] (first element) is associated with.
- Note n[0] and ptr_to_int[0] refer to the same memory and n[1] and ptr_to_int[1] refer to the same memory and so on.

PFI lecture POINTERS

- Pointer variable can be used to refer to the memory location to which the pointer points. This is called dereferencing the pointer. Prefix "*" operator to a pointer.
- Examples:
  - Pointer to int

```
int * ptr_to_int;
int n=7;
ptr_to_int = &n;
n = n * *ptr_to_int; // *ptr_to_int is dereferencing
*ptr_to_int = 8; // the value of n is 8 now
```

PFI lecture POINTERS

- Pointer variables are like other variables and we may copy their values via assignment statements.
- Examples:
  - Pointer to int

```
int * ptr_to_int;
int* ptr;
int n=7;
ptr_to_int = &n;
n = n * *ptr_to_int; // *ptr_to_int is dereferencing
*ptr_to_int = 8; // the value of n is 8 now
ptr = ptr_to_int;
*ptr = 10  // the value of n is 10, so is *ptr_to_int
```

PFI lecture POINTERS

- Pointer arithmetic: we may add (or subtract) an integer to a pointer and we may subtract two pointers of the same type and pointing to the same memory block. Addresses are manipulated in pointer arithmeic.
- Adding one to a pointer advances the pointer by the number of bytes needed for the type to which the pointer points.
- Examples:
  - Pointer to int

```
int * ptr_to_int;
int* ptr;
int n[7] ={1,3,5,7,9,11,13};
ptr_to_int = n;
ptr = ptr_to_int +1;
*ptr = 8; // {1,8,5,7,9,11,13}
ptr[2] = 0;// {1,8,5,0,9,11,13}
*(ptr +4) = 10  // {1,8,5,0,9,10,13}
```

PFI lecture POINTERS

- Pointer variable can be used to refer to the memory location pointed to by the pointer. This is called dereferencing the pointer. Another way to dereferencing is to use array index notation.
- Examples:
  - Pointer to int

```
int * ptr_to_int;
int n=7;
ptr_to_int = &n;
n = n * ptr_to_int[0]; // ptr_to_int[0] is dereferencing
ptr_to_int[0] = 8; // the value of n is 8 now
```

PFI lecture POINTERS

- How do we assign values a pointer variable? Or where and how do we get the address of another memory location? The **new** operator and dynamic memory allocation
- Examples:
  - Pointer to int

  int * ptr_to_int;

  ptr_to_int = new int;

  int * ptr = new int[10]; // 10 int one after another, array!
- The only way to access the memory allocated dynamically is to use the value stored in the pointer. If the value in the pointe is lost, we cannot access the memroy (memory leak) any more.

Why do we need a pointer?

To dynamically allocate memory to store information. Note that when we declare an array, the size or capacity is fixed. If the size is too small, we cannot hold all the data. If the size is too large, we waste space.

To build linked structures, where we may go from object to another object, where objects are not stored one after another in memory as in the case of arrays.

Why does a variable need a type?

Type specifies the size of memory and how to interpret the binary value (0 and 1 pattern)

For example, 01000001 binary value, is 'A' for char type, true for bool type, and 65 for int type.

- With pointers, we have another category of types. The sizes of all pointer types are the same, sufficient to hold a memory address. The interpretation also depends on the type that the pointer points to. See dereferencing and pointer arithmetic slides.

How do we get a value to a pointer variable in our code?

- Use &var to get the address of variable var
  - See earlier slides
- Use an array name
  - See earlier slides
- Use new operator, dynamically allocate memory
  - See earlier slides
- Get the value from another pointer variable
  - The other pointer variable's value is obtained by using the 3 methods above or via copy
  - Use assignment or parameter passing to get value from another pointer variable.

An example applying basic concepts

```
struct Node{ // struct is a public class, all members are public
    int data;
    Node * ptr; // pointer to Node type, class introduce a type
};
// main program
Node * p = new Node; // dynamic memory allocation
Node * save = p;
(*p).data = 7; // “dot” notation for attribute or method
(*p).ptr = new Node;
p = (*p).ptr; // note the type of p and p.ptr are the same
(*p).data = 77;
(*p).ptr = new Node;
p = (*p).ptr;
(*p).data = 777;
(*p).ptr = NULL;  // NULL is a special pointer value that do not point to
    anything
```

An example using basic concepts (continuation)

```
p = save;
while ( p != NULL){
                cout << (*p).data << endl;
                p=(*p).ptr;
}
p = save;
while ( p != NULL){
                 (*p).data = (*p).data+(*p).data;
                p=(*p).ptr;
}
p = save;
while ( p != NULL){
                cout << (*p).data << endl;
                p=(*p).ptr;
}
```

An example using basic concepts (continuation)

```
p = save;
while ( p != NULL){
        save = p;
        p=(*p).ptr;
        delete save;
}
// we have returned the memory dynamically allocated.
```

- “delete” operator frees the space pointed to by a pointer
- delete ptr; // frees the object pointed by ptr
- delete [] ptr; // frees the chuck of memory (array) pointer to by ptr

How do we “see” the value of a pointer variable?

- In debugging and in figuring out what is going on with our program, or simply gain a deeper understanding of memory structure, we often need to know what are the values in our pointer variables.
- cout

```
int* ptr;
// code to get value into ptr
cout << “The value of ptr is ”  <<  ptr;  // in hex
```