Pointers & Strings.
Part 2.

Week 9

Assylbek Jumagaliyev

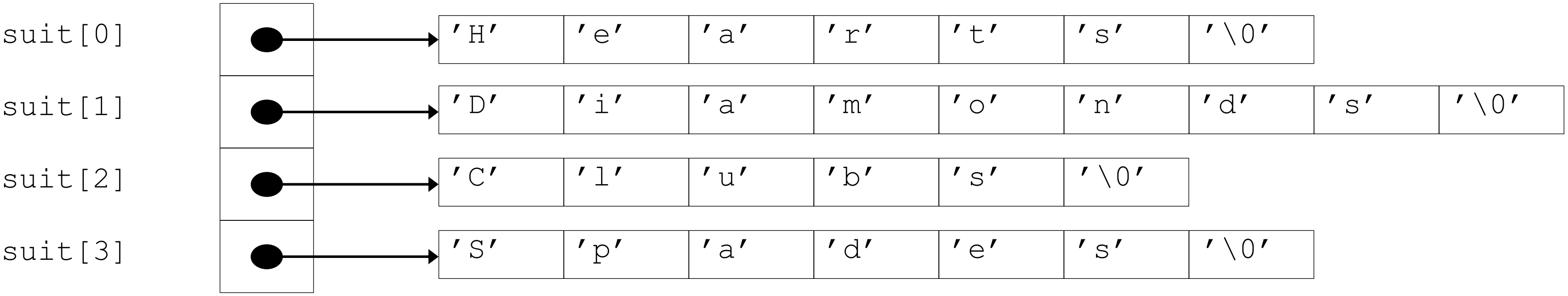
a.jumagaliyev@iitu.kz

Arrays of Pointers

- Arrays can contain pointers
  - Commonly used to store array of strings

```
char *suit[ 4 ] = {"Hearts", "Diamonds",
                   "Clubs", "Spades" };
```

  - Each element of **suit** points to **`char *`** (a string)
  - Array does not store strings, only pointers to strings



suit[0] 'H' 'e' 'a' 'r' 't' 's' '\0'

suit[1] 'D' 'i' 'a' 'm' 'o' 'n' 'd' 's' '\0'

suit[2] 'C' 'l' 'u' 'b' 's' '\0'

suit[3] 'S' 'p' 'a' 'd' 'e' 's' '\0'

  - **`suit`** array has fixed size, but strings can be of any size

Function Pointers

- Pointers to functions
  - Contain address of function
  - Similar to how array name is address of first element
  - Function name is starting address of code that defines function
- Function pointers can be
  - Passed to functions
  - Returned from functions
  - Stored in arrays
  - Assigned to other function pointers

Function Pointers

- Calling functions using pointers
  - Assume parameter:
    - **`bool ( *compare ) ( int, int )`**
  - Execute function with either
    - **`( *compare ) ( int1, int2 )`**
      - Dereference pointer to function to execute

    OR

    - **`compare( int1, int2 )`**
      - Could be confusing
        - » User may think **`compare`** name of actual function in program

```cpp
// Fig. 5.25: fig05_25.cpp
// Multipurpose sorting program using function pointers.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

#include <iomanip>

using std::setw;

// prototypes
void bubble( int [], const int, bool (*)( int, int ) );
void swap( int * const, int * const );
bool ascending( int, int );
bool descending( int, int );

int main()
{
   const int arraySize = 10;
   int order;
   int counter;
   int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

```

Parameter is pointer to function that receives two integer parameters and returns `bool` result.

```
   cout << "Enter 1 to sort in ascending order,\n"
        << "Enter 2 to sort in descending order: ";
   cin >> order;
   cout << "\nData items in original order\n";

   // output original array
   for ( counter = 0; counter < arraySize; counter++ )
      cout << setw( 4 ) << a[ counter ];

   // sort array in ascending order; pass function ascending
   // as an argument to specify ascending sorting order
   if ( order == 1 ) {
      bubble( a, arraySize, ascending );
      cout << "\nData items in ascending order\n";
   }

   // sort array in descending order; pass function descending
   // as an agrument to specify descending sorting order
   else {
      bubble( a, arraySize, descending );
      cout << "\nData items in descending order\n";
   }

   // output sorted array
   for ( counter = 0; counter < arraySize; counter++ )
      cout << setw( 4 ) << a[ counter ];

   cout << endl;

   return 0;  // indicates successful termination

} // end main

// multipurpose bubble sort; parameter compare is a pointer to
// the comparison function that determines sorting order
void bubble( int work[], const int size,
         bool (*compare)( int, int ) )
{
   // loop to control passes
   for ( int pass = 1; pass < size; pass++ )

      // loop to control number of comparisons per pass
      for ( int count = 0; count < size - 1; count++ )

         // if adjacent elements are out of order, swap them
         if ( (*compare)( work[ count ], work[ count + 1 ] ) )
            swap( &work[ count ], &work[ count + 1 ] );
```

`compare` is pointer to function that receives two integer parameters and returns `bool` result.

Parentheses necessary to indicate pointer to function

Call passed function `compare`; dereference pointer to execute function.

```
} // end function bubble

// swap values at memory locations to which
// element1Ptr and element2Ptr point
void swap( int * const element1Ptr, int * const element2Ptr )
{
   int hold = *element1Ptr;
   *element1Ptr = *element2Ptr;
   *element2Ptr = hold;

} // end function swap

// determine whether elements are out of order
// for an ascending order sort
bool ascending( int a, int b )
{
   return b < a;   // swap if b is less than a

} // end function ascending
```

```cpp
// determine whether elements are out of order
// for a descending order sort
bool descending( int a, int b )
{
   return b > a;   // swap if b is greater than a

} // end function descending
```

```
Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 1

Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
```

```
Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 2

Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in descending order
  89  68  45  37  12  10   8   6   4   2
```

Function Pointers

- Arrays of pointers to functions
  - Menu-driven systems
  - Pointers to each function stored in array of pointers to functions
    - All functions must have same return type and same parameter types
  - Menu choice → subscript into array of function pointers

```cpp
// Fig. 5.26: fig05_26.cpp
// Demonstrating an array of pointers to functions.
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

// function prototypes
void function1( int );
void function2( int );
void function3( int );

int main()
{
   // initialize array of 3 pointers to functions that each
   // take an int argument and return void
   void (*f[ 3 ])( int ) = { function1, function2, function3 };

   int choice;

   cout << "Enter a number between 0 and 2, 3 to end: ";
   cin >> choice;

```

Array initialized with names of three functions; function names are pointers.

```
   // process user's choice
   while ( choice >= 0 && choice < 3 ) {

      // invoke function at location choice in array f
      // and pass choice as an argument
      (*f[ choice ])( choice );

      cout << "Enter a number between 0 and 2, 3 to end: ";
      cin >> choice;
   }

   cout << "Program execution completed." << endl;

   return 0;  // indicates successful termination

} // end main

void function1( int a )
{
   cout << "You entered " << a
        << " so function1 was called\n\n";

} // end function1

```

Call chosen function by dereferencing corresponding element in array.

```
void function2( int b )
{
   cout << "You entered " << b
        << " so function2 was called\n\n";

} // end function2

void function3( int c )
{
   cout << "You entered " << c
        << " so function3 was called\n\n";

} // end function3
```

```
Enter a number between 0 and 2, 3 to end: 0
You entered 0 so function1 was called

Enter a number between 0 and 2, 3 to end: 1
You entered 1 so function2 was called

Enter a number between 0 and 2, 3 to end: 2
You entered 2 so function3 was called

Enter a number between 0 and 2, 3 to end: 3
Program execution completed.
```

Fundamentals of Characters and Strings

- Character constant
  - Integer value represented as character in single quotes
  - **`'z'`** is integer value of **`z`**
    - **`122`** in ASCII
- String
  - Series of characters treated as single unit
  - Can include letters, digits, special characters **`+`**, **`-`**, **`*`** ...
  - String literal (string constants)
    - Enclosed in double quotes, for example:

      `"I like C++"`
  - Array of characters, ends with null character **`'\0'`**
  - String is constant pointer
    - Pointer to string’s first character
      - Like arrays

Fundamentals of Characters and Strings

- String assignment
  - Character array
    - **`char color[] = "blue";`**
      - Creates 5 element **`char`** array **`color`**
        - » last element is **`'\0'`**
  - Variable of type **`char *`**
    - **`char *colorPtr = "blue";`**
      - Creates pointer **`colorPtr`** to letter **`b`** in string “**`blue`**”
        - » “**`blue`**” somewhere in memory
  - Alternative for character array
    - **`char color[] = { ‘b’, ‘l’, ‘u’, ‘e’, ‘\0’ };`**

Fundamentals of Characters and Strings

- Reading strings
  - Assign input to character array **`word[ 20 ]`**

    ```
    cin >> word
    ```

    - Reads characters until whitespace or EOF
    - String could exceed array size

    ```
    cin >> setw( 20 ) >> word;
    ```

    - Reads 19 characters (space reserved for **`'\0'`**)

Fundamentals of Characters and Strings

- **`cin.getline`**
  - Read line of text
  - **`cin.getline( array, size, delimiter );`**
  - Copies input into specified **`array`** until either
    - One less than **`size`** is reached
    - **`delimiter`** character is input
  - Example

```
char sentence[ 80 ];
cin.getline( sentence, 80, '\n' );
```

String Manipulation Functions of the String-handling Library

- String handling library **`<cstring>`** provides functions to
  - Manipulate string data
  - Compare strings
  - Search strings for characters and other strings
  - Tokenize strings (separate strings into logical pieces)

String Manipulation Functions of the String-handling Library

`char *strcpy( char *s1, const char *s2 );`	Copies the string `s2` into the character array `s1`. The value of `s1` is returned.
`char *strncpy( char *s1, const char *s2, size_t n );`	Copies at most `n` characters of the string `s2` into the character array `s1`. The value of `s1` is returned.
`char *strcat( char *s1, const char *s2 );`	Appends the string `s2` to the string `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.
`char *strncat( char *s1, const char *s2, size_t n );`	Appends at most `n` characters of string `s2` to string `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.
`int strcmp( const char *s1, const char *s2 );`	Compares the string `s1` with the string `s2`. The function returns a value of zero, less than zero or greater than zero if `s1` is equal to, less than or greater than `s2`, respectively.

String Manipulation Functions of the String-handling Library

`int strncmp( const char *s1, const char *s2, size_t n );`	Compares up to **`n`** characters of the string **`s1`** with the string **`s2`**. The function returns zero, less than zero or greater than zero if **`s1`** is equal to, less than or greater than **`s2`**, respectively.
`char *strtok( char *s1, const char *s2 );`	A sequence of calls to **`strtok`** breaks string **`s1`** into “tokens”—logical pieces such as words in a line of text—delimited by characters contained in string **`s2`**. The first call contains **`s1`** as the first argument, and subsequent calls to continue tokenizing the same string contain **`NULL`** as the first argument. A pointer to the current to-ken is returned by each call. If there are no more tokens when the function is called, **`NULL`** is returned.
`size_t strlen( const char *s );`	Determines the length of string **`s`**. The number of characters preceding the terminating null character is returned.

String Manipulation Functions of the String-handling Library

- Copying strings
  - **`char *strcpy( char *s1, const char *s2 )`**
    - Copies second argument into first argument
      - First argument must be large enough to store string and terminating null character
  - **`char *strncpy( char *s1, const char *s2, size_t n )`**
    - Specifies number of characters to be copied from string into array
    - Does not necessarily copy terminating null character

```cpp
// Fig. 5.28: fig05_28.cpp
// Using strcpy and strncpy.
#include <iostream>

using std::cout;
using std::endl;

#include <cstring>  // prototypes for strcpy and strncpy

int main()
{
   char x[] = "Happy Birthday to You";
   char y[ 25 ];
   char z[ 15 ];

   strcpy( y, x );  // copy contents of x into y

   cout << "The string in array x is: " << x
        << "\nThe string in array y is: " << y << '\n';

   // copy first 14 characters of x into z
   strncpy( z, x, 14 );  // does not copy null character
   z[ 14 ] = '\0';       // append '\0' to z's contents

   cout << "The string in array z is: " << z << endl;
```

<cstring> contains prototypes for strcpy and strncpy.

Copy entire string in array x into array y.

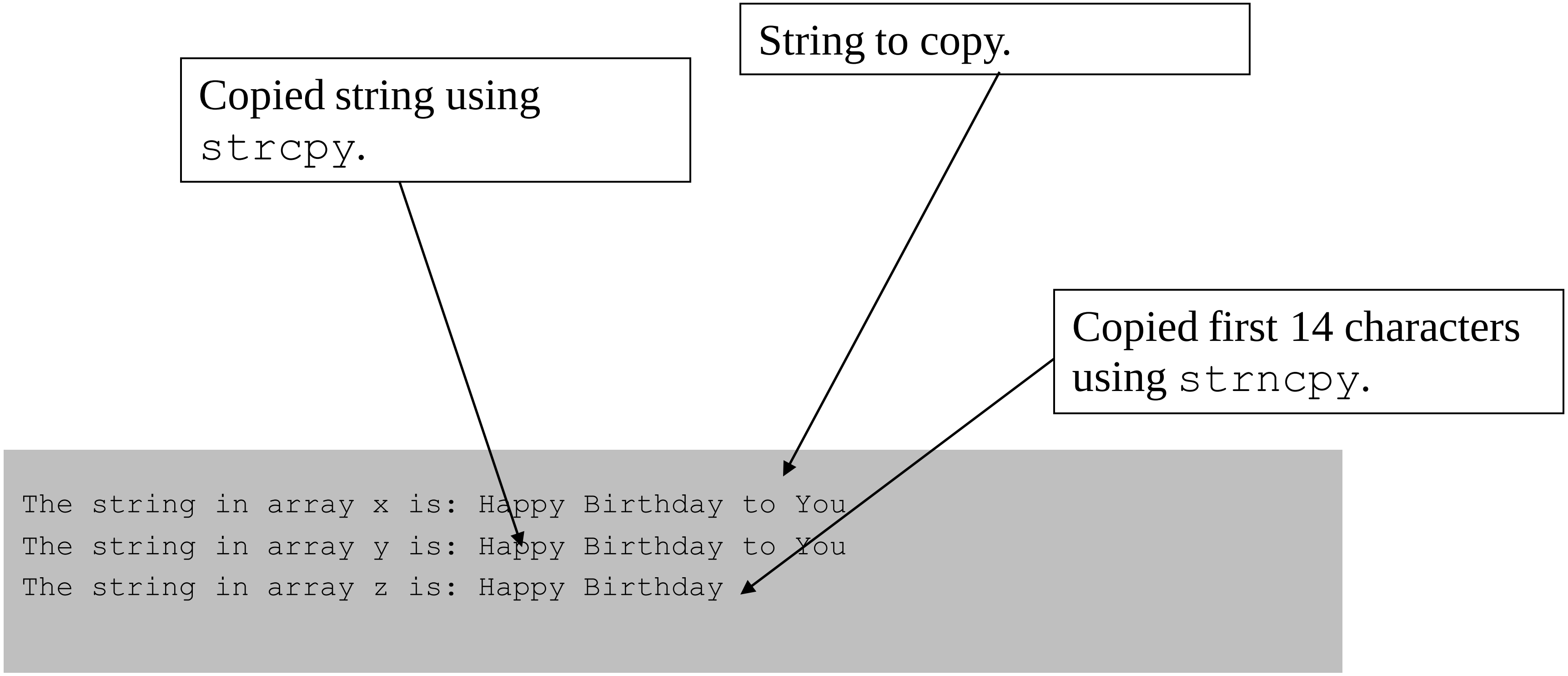
Copy first 14 characters of array x into array y. Note that this does not write terminating null character.

Append terminating null character.

```cpp

   return 0;  // indicates successful termination

} // end main
```



```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

Copied string using `strcpy`.

String to copy.

Copied first 14 characters using `strncpy`.

String Manipulation Functions of the String-handling Library

- Concatenating strings
  - `char *strcat( char *s1, const char *s2 )`
    - Appends second argument to first argument
    - First character of second argument replaces null character terminating first argument
    - Ensure first argument large enough to store concatenated result and null character
  - `char *strncat( char *s1, const char *s2, size_t n )`
    - Appends specified number of characters from second argument to first argument
    - Appends terminating null character to result

```cpp
// Fig. 5.29: fig05_29.cpp
// Using strcat and strncat.
#include <iostream>

using std::cout;
using std::endl;

#include <cstring>   // prototypes for strcat and strncat

int main()
{
  char s1[ 20 ] = "Happy ";
  char s2[] = "New Year ";
  char s3[ 40 ] = "";

  cout << "s1 = " << s1 << "\ns2 = " << s2;

  strcat( s1, s2 );  // concatenate s2 to s1

  cout << "\n\nAfter strcat(s1, s2):\ns1 = " << s1
      << "\ns2 = " << s2;

  // concatenate first 6 characters of s1 to s3
  strncat( s3, s1, 6 );  // places '\0' after last character

```

`<cstring>` contains prototypes for `strcat` and `strncat`.

Append `s2` to `s1`.

Append first 6 characters of `s1` to `s3`.

```cpp
   cout << "\n\nAfter strncat(s3, s1, 6):\ns1 = " << s1
      << "\ns3 = " << s3;

   strcat( s3, s1 );  // concatenate s1 to s3
   cout << "\n\nAfter strcat(s3, s1):\ns1 = " << s1
      << "\ns3 = " << s3 << endl;

   return 0;  // indicates successful termination

} // end main
```

Append `s1` to `s3`.

```
s1 = Happy
s2 = New Year

After strcat(s1, s2):
s1 = Happy New Year
s2 = New Year

After strncat(s3, s1, 6):
s1 = Happy New Year
s3 = Happy

After strcat(s3, s1):
s1 = Happy New Year
s3 = Happy Happy New Year
```

String Manipulation Functions of the String-handling Library

- Comparing strings
  - Characters represented as numeric codes
    - Strings compared using numeric codes
  - Character codes / character sets
    - ASCII
      - “American Standard Code for Information Interchage”
    - EBCDIC
      - “Extended Binary Coded Decimal Interchange Code”

String Manipulation Functions of the String-handling Library

- Comparing strings
  - **`int strcmp( const char *s1, const char *s2 )`**
    - Compares character by character
    - Returns
      - Zero if strings equal
      - Negative value if first string less than second string
      - Positive value if first string greater than second string
  - **`int strncmp( const char *s1, const char *s2, size_t n )`**
    - Compares up to specified number of characters
    - Stops comparing if reaches null character in one of arguments

```
// Fig. 5.30: fig05_30.cpp
// Using strcmp and strncmp.
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

#include <cstring>  // prototypes for strcmp and strncmp

int main()
{
   char *s1 = "Happy New Year";
   char *s2 = "Happy New Year";
   char *s3 = "Happy Holidays";

   cout << "s1 = " << s1 << "\ns2 = " << s2
      << "\ns3 = " << s3 << "\n\nstrcmp(s1, s2) = "
      << setw( 2 ) << strcmp( s1, s2 )
      << "\nstrcmp(s1, s3) = " << setw( 2 )
      << strcmp( s1, s3 ) << "\nstrcmp(s3, s1) = "
      << setw( 2 ) << strcmp( s3, s1 );
```

`<cstring>` contains prototypes for `strcmp` and `strncmp`.

Compare `s1` and `s2`.

Compare `s1` and `s3`.

Compare `s3` and `s1`.

```cpp

   cout << "\n\nstrncmp(s1, s3, 6) = " << setw( 2 )
      << strncmp( s1, s3, 6 ) << "\nstrncmp(s1, s3, 7) = "
      << setw( 2 ) << strncmp( s1, s3, 7 )
      << "\nstrncmp(s3, s1, 7) = "
      << setw( 2 ) << strncmp( s3, s1, 7 ) << endl;

   return 0;  // indicates successful termination

} // end main
```

Compare up to 6 characters of `s1` and `s3`.

Compare up to 7 characters of `s1` and `s3`.

Compare up to 7 characters of `s3` and `s1`.

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  1
strncmp(s3, s1, 7) = -1
```

String Manipulation Functions of the String-handling Library

- Tokenizing
  - Breaking strings into tokens, separated by delimiting characters
  - Tokens usually logical units, such as words (separated by spaces)
  - **`"This is my string"`** has 4 word tokens (separated by spaces)
  - **`char *strtok( char *s1, const char *s2 )`**
    - Multiple calls required
      - First call contains two arguments, string to be tokenized and string containing delimiting characters
        - » Finds next delimiting character and replaces with null character
      - Subsequent calls continue tokenizing
        - » Call with first argument **`NULL`**

```
// Fig. 5.31: fig05_31.cpp
// Using strtok.
#include <iostream>

using std::cout;
using std::endl;

#include <cstring>  // prototype for strtok

int main()
{
  char sentence[] = "This is a sentence with 7 tokens";
  char *tokenPtr;

  cout << "The string to be tokenized is:\n" << sentence
       << "\n\nThe tokens are:\n\n";

  // begin tokenization of sentence
  tokenPtr = strtok( sentence, " " );

```

`<cstring>` contains prototype for `strtok`.

First call to `strtok` begins tokenization.

```
   // continue tokenizing sentence until tokenPtr becomes NULL
   while ( tokenPtr != NULL ) {
      cout << tokenPtr << '\n';
      tokenPtr = strtok( NULL, " " );  // get next token

   } // end while

   cout << "\nAfter strtok, sentence = " << sentence << endl;

   return 0;  // indicates successful termination

} // end main
```

Subsequent calls to `strtok` with `NULL` as first argument to indicate continuation.

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:

This
is
a
sentence
with
7
tokens

After strtok, sentence = This
```

String Manipulation Functions of the String-handling Library

- Determining string lengths
  - `size_t strlen( const char *s )`
    - Returns number of characters in string
      - Terminating null character not included in length

```cpp
// Fig. 5.32: fig05_32.cpp
// Using strlen.
#include <iostream>

using std::cout;
using std::endl;

#include <cstring>  // prototype for strlen

int main()
{
  char *string1 = "abcdefghijklmnopqrstuvwxyz";
  char *string2 = "four";
  char *string3 = "Boston";

  cout << "The length of \"" << string1
      << "\" is " << strlen( string1 )
      << "\nThe length of \"" << string2
      << "\" is " << strlen( string2 )
      << "\nThe length of \"" << string3
      << "\" is " << strlen( string3 ) << endl;

  return 0;  // indicates successful termination

} // end main
```

`<cstring>` contains prototype for `strlen`.

Using `strlen` to determine length of strings.

- The length of "abcdefghijklmnopqrstuvwxyz" is 26
- The length of "four" is 4
- The length of "Boston" is 6

Dynamic variables

- Since a pointer can be used to refer to a variable, your program can manipulate variables even if the variables have no identifiers to name them.
- The operator **new** can be used to create variables that have no identifiers to serve as their names. These nameless variables are referred to via pointers.
- Example:
  - **p1 = new int;**
- This new, nameless variable can be referred to as *p1 (that is, as the variable pointed to by p1)

Dynamic variables

- Variables that are created using the new operator are called **dynamically allocated variables** or simply **dynamic variables**
- The **delete** operator eliminates a dynamic variable and returns the memory that the dynamic variable occupied to the freestore.
- For example, the following eliminates the dynamic variable pointed to by the pointer variable p:
  - **delete p;**
- After a call to delete, the value of the pointer variable, like p above, is undefined.

```
//Program to demonstrate pointers and dynamic variables.
#include <iostream>
using std::cout;
using std::endl;
int main( )
{
  int *p1, *p2;
  p1 = new int;
 *p1 = 42;
  p2 = p1;
  cout << "*p1 == " << *p1 << endl;
  cout << "*p2 == " << *p2 << endl;
  *p2 = 53;
  cout << "*p1 == " << *p1 << endl;
  cout << "*p2 == " << *p2 << endl;
  p1 = new int;
  *p1 = 88;
  cout << "*p1 == " << *p1 << endl;
  cout << "*p2 == " << *p2 << endl;
  cout << "Hope you got the point of this example!\n";
return 0;
}
```

DEFINE POINTER TYPES

- You can define a pointer type name so that pointer variables can be declared like other variables without the need to place an asterisk in front of each pointer variable.

```
typedef int* IntPtr;
```

- Thus, the following two pointer variable declarations are equivalent:

```
    IntPtr p;
```

and

```
    int *p;
```

DEFINE POINTER TYPES

TYPE DEFINITIONS

- You can assign a name to a type definition.
- **keyword typedef**.
  - normally placed
    - outside the body of the main part of your program
    - outside the body of other functions
    - typically near the start of a file. T

SYNTAX

```
typedef Known_Type_Definition New_Type_Name;
```

EXAMPLE

```
typedef int* IntPtr;
IntPtr pointer1, pointer2;
```

Dynamic Arrays

- Dynamically allocated arrays are created using the ***new*** operator.
- Example:

```
Typedef  double*  DoublePtr;
DoublePtr  d;
d = new double[10];
```

```
//Searches a list of numbers entered at the keyboard.
#include <iostream>

using std::cin;
using std::cout;

typedef int* IntPtr;

void fillArray(int a[], int size);

//Precondition: size is the size of the array a.
//Postcondition: a[0] through a[size-1] have been
//filled with values read from the keyboard.

int search(int a[], int size, int target);
//Precondition: size is the size of the array a.
//The array elements a[0] through a[size-1] have values.
//If target is in the array, returns the first index of target.
//If target is not in the array, returns -1.
```

```
int main( )
{
  cout << "This program searches a list of numbers.\n";
  int arraySize;
  cout << "How many numbers will be on the list? ";
  cin >> arraySize;
  IntPtr a;
  a = new int[arraySize];
  fillArray(a, arraySize);
  int target;
  cout << "Enter a value to search for: ";
  cin >> target;
  int location = search(a, arraySize, target);
  if (location == -1)
    cout << target << " is not in the array.\n";
  else
    cout << target << " is element " << location << " in the array.\n";

  delete [] a;

return 0;
}
```

```
//Uses the library <iostream>:
void fillArray(int a[], int size)
{
  cout << "Enter " << size << " integers.\n";
  for (int index = 0; index < size; index++)
    cin >> a[index];
}

int search(int a[], int size, int target)
{
  int index = 0;
  while ((a[index] != target) && (index < size))
    index++;
  if (index == size)//if target is not in a.
    index = -1;
  return index;
}
```

Dynamic Arrays

- The ***delete*** statement for a dynamically allocated array:

  delete [ ] a;

- The square brackets tell C++ that a dynamically allocated array variable is being eliminated.

HOW TO USE A DYNAMIC ARRAY

- *Define a pointer type*:

  typedef double* DoubleArrayPtr;

- *Declare a pointer variable*:

  DoubleArrayPtr a;

- (Alternatively, without a defined pointer type, use double *a;).

HOW TO USE A DYNAMIC ARRAY

- *Call new*: Create a dynamic array using the new operator:

  a = new double[arraySize];

- The size of the dynamic array is given in square brackets.
- *Use like an ordinary array*: The pointer variable, such as a, is used just like an ordinary array.

HOW TO USE A DYNAMIC ARRAY

- *Call delete [ ]*

- For example:

  delete [ ] a;

```cpp
//A Two-Dimensional Dynamic Array
#include <iostream>

using std::cin;
using std::cout;
using std::endl;

typedef int* IntArrayPtr;

int main( )
{
  int d1, d2;
  cout << "Enter the row and column dimensions of the array:\n";
  cin >> d1 >> d2;

  IntArrayPtr *m = new IntArrayPtr[d1];
  int i, j;

  for (i = 0; i < d1; i++)
    m[i] = new int[d2];
  //m is now a d1-by-d2 array.
```

```cpp
 cout << "Enter " << d1 << " rows of "
       << d2 << " integers each:\n";

  for (i = 0; i < d1; i++)
    for (j = 0; j < d2; j++)
      cin >> m[i][j];
 for (i = 0; i < d1; i++)
  {
    for (j = 0; j < d2; j++)
      cout << m[i][j] << " ";
    cout << endl;
  }

  for (i = 0; i < d1; i++)
    delete[] m[i];

  delete[] m;

  return 0;
}
```

Readings:

- **C++ How to Program,** By H. M. Deitel
  - Chapter 8. Pointers and Pointer-Based Strings
    - Parts 8.10 – 8.13
- **C++ beginner's guide** by Schildt
  - Chapter 4 Arrays, Strings and Pointers
    - Parts 4.7 – 4.12
- **Absolute C++** by Savitch
  - Chapter 10 Pointers and Dynamic Arrays
    - Part 10.2

Thanks for Your Attention!