// InitStrings: Initializing and Printing Strings

#include <iostream> // for cout
#include <cstring> // for strlen()

using namespace std;

int main()
{
    char name[10] = "Zelda"; // name is an address constant
    char carray[] = "What about you?"; // carray is an address constant
    char* pchar = "Not much to me."; // pchar is a variable

    cout << name << " , " << carray << " " << pchar << "\n\n";

    int index;
    for (index = 0; index < strlen(carray); index++)
        cout << carray[index]; // Array notation
    cout << "\t\t";

    char* pc;
    for (pc = carray; pc < carray + strlen(carray); pc++)
        cout << *pc; // Pointer notation
    cout << "\n\n";

    pc = pchar; // Preserve address in pchar
    while (*pc != '\0')
    {
        cout << *pc; // Pointer notation
        pc++;
    } 
    cout << "\t\t";

    // Shorten the above loop to the following loop
    pc = pchar; // Preserve address in pchar
    while (*pc)
        cout << *pc; // Use *pc, then increment pc
    cout << "\t\t";

    while (*pchar) // Don't preserve address in pchar
        cout << *pchar++;
    cout << "\n\n";

    pchar = carray; // Previous loop changed the value of pchar
    cout << carray << " " << pchar << "\n";

    return 0;
}

Output:
Zelda, What about you? Not much to me. (14)
What about you? What about you? (18, 23)
Not much to me. Not much to me. Not much to me. (29, 37, 41)
What about you? What about you? (45)
1.  
```c
char name[10] = "Zelda";  // or char name[10] = {'Z', 'e', 'l', 'd', 'a', '\0'};
```

Local array on the stack; `name` is an address constant.

![Z e l d a \0\0\0\0\0
0 1 2 3 4 5 6 7 8 9](image)

Local variables (arrays too) contain GARBAGE if not initialized.

Local arrays: Partially initialized; uninitialized elements are set to 0

0 in char form is the null character.

2.  
```c
char carray[ ] = "What about you?";
```

"incomplete type" - declaration is completed by the initializer

Local array on the stack; `carray` is an address constant.

![W h a t ^a b o u t ^y o u ? \0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15](image)

Compiler inserts the null character in the C string constant.

3.  
```c
char* pchar = "Not much to me.";
```

String is stored in permanent storage; `pchar` is a pointer variable on the stack and initially points to the string.

![N o t ^m u c h ^t o ^m e . \0](image)

4. Ways to define a string:

   a. string constant
   b. char array
   c. char pointer
   d. array of character strings
// strcatFun: string concatenation - strcat()

/* char* strcat(char* string1, const char* string2);
strcat() concatenates a copy of string2 to string1 and
returns the address of string1. string2 is not changed.

Usually, string1 will be char array that should
be large enough to hold the concatenated result. */

#include <iostream>    // cout
#include <cstring>     // strcat(), strlen()}
using namespace std;
const int SIZE = 81;

int main()
{
    char first_message[SIZE] = "Stacks are swell";
    char second_message[SIZE] = "Linked lists are lovely";
    cout << "Before CONCATENATION:
    " << first_message << '
';
    cout << second_message << '
';
    if (strlen(first_message) + strlen(second_message) + 1 <= SIZE)
    {
        strcat(first_message, second_message);
        cout << "After CONCATENATION:
        " << first_message << '
';
        cout << second_message << '
';
    }
    else
    {
        cout << "CONCATENATION FAILED:
        " << "Not enough storage for concatenated strings!";
        return 0;
    }

    Run 1 with SIZE equal to 81:
Before CONCATENATION:
Stacks are swell
Linked lists are lovely
After CONCATENATION:
Stacks are swellLinked lists are lovely
Linked lists are lovely

Run 2 with SIZE equal to 25:
Before CONCATENATION:
Stacks are swell
Linked lists are lovely
CONCATENATION FAILED:
Not enough storage for concatenated strings!
Possible Definitions for `strcat()`:

**Array Notation:**

```c
char* strcat(char* s1, const char* s2)
{
    int i, j;
    for (i = 0; s1[i] != '\0'; i++)
        continue;
    for (j = 0; s2[j] != '\0'; j++, i++)
        s1[i] = s2[j];
    s1[i] = '\0';
    return s1;
}
```

**Pointer Notation:**

```c
char* strcat(char* s1, const char* s2)
{
    char* s;
    for (s = s1; *s; ++s)
        continue;
    for (; *s = *s2; ++s, ++s2)
        continue;
    return s1;
}
```
// strcmpFun: string compare - strcmp()

/* int strcmp(const char* string1, const char* string2);

strcmp() lexicographically compares two strings and returns an integer based on the comparison:

  Return value
  that is:

less than 0  means string1 is less than  string2
0            means string1 is equal to    string2
greater than 0 means string1 is greater than string2 */

#include <iostream>  // cout, getline()
#include <cstring>   // strcmp()

using namespace std;

const int SIZE = 81;

int main()
{
    char password[SIZE];
    cout << "Enter your password: ";
    cin.getline(password, SIZE);
    while (strcmp(password, "secret")) // strcmp(password, "secret") != 0
    {
        cout << "Invalid password - try again: ";
        cin.getline(password, SIZE);
    }
    cout << "Valid password. Now you will see some secret information."
    cout << "A == A: " << strcmp("A", "A") << '\n';
    cout << "A < B: " << strcmp("A", "B") << '\n';
    cout << "B > A: " << strcmp("B", "A") << '\n';
    cout << "ALFRED < ALICE: " << strcmp("ALFRED", "ALICE") << '\n';
    cout << "ALICE > ALFRED: " << strcmp("ALICE", "ALFRED") << '\n';
    cout << "ROBERTS < ROBERTSON: " << strcmp("ROBERTS", "ROBERTSON") << '\n';
    cout << "ROBERTSON > ROBERTS: " << strcmp("ROBERTSON", "ROBERTS") << '\n';
    return 0;
}

Output:
Enter your password: tree
Invalid password - try again: cool
Invalid password - try again: secret
Valid password. Now you will see some secret information.
A == A: 0
A < B: -1
B > A: 1
ALFRED < ALICE: -1
ALICE > ALFRED: 1
ROBERTS < ROBERTSON: -1
ROBERTSON > ROBERTS: 1
Possible Definition for `strcmp()`:

```c
int strcmp(const char* s1, const char* s2)
{
    int i;
    for (i = 0; s1[i] == s2[i]; i++)
        if (s1[i] == '\0')
            return 0;
    return (s1[i] < s2[i] ? -1 : 1);
}
```
// strcpyFun: string copy - strcpy()

/* char* strcpy(char* string1, const char* string2);

   strcpy() copies string2 to string1. string2, the source, can be
   any pointer to a string, that is, an array name, pointer-to-char,
   or a string constant. string1, the target, usually is an array.

   It is the programmer's responsibility to ensure that the target
   array is large enough to hold the source string.

   strcpy() returns its first argument, that is, the address of the
   target. Neither argument must point to the beginning of the string. */

#include <iostream> // cout
#include <cstring> // strcpy()

using namespace std;

const int SIZE = 81;

int main()
{
    char first_message[SIZE] = "Bye!";
    char second_message[SIZE] = "Hi There";
    char* pc;

cout << "Before assignment:
first_message contains: " << static_cast<void*> (first_message) << " pc contains: " << static_cast<void*> (pc) << "\n";

pc = first_message; // Copies address, not the string

cout << "After assignment:
first_message contains: " << static_cast<void*> (first_message) << " pc contains: " << static_cast<void*> (pc) << "\n";

cout << first_message << "\n"
    << pc << "\n\n";

cout << second_message << "\n";
strcpy(second_message, first_message); // Copy array to array

cout << second_message << "\n";

strcpy(second_message, "Oh no!"); // Copy string constant to array

cout << second_message << "\n";

strcpy(second_message, pc); // Copy pointed-to string to array

cout << second_message << "\n";

strcpy(second_message + 3, first_message);

cout << second_message << "\n";

strcpy(first_message, "Joe Montana");

cout << first_message << "\n";

pc = strcpy(second_message + 6, first_message + 4);

cout << pc << "\n";

cout << second_message << "\n";
    return 0;
}
Output:
Before assignment:
first_message contains: 0012FF2C pc contains: 0012FEC0
After assignment:
first_message contains: 0012FF2C pc contains: 0012FF2C
Bye!  
(line 37)
Bye!

Hi There  
(line 40)
Bye!  
(line 42)
Oh no!  
(line 45)
Bye!  
(line 48)
ByeBye!  
(line 51)
Joe Montana  
(line 54)
Montana  
(line 56)
ByeByeMontana  
(line 57)

Contents of the char arrays:

<table>
<thead>
<tr>
<th>first_message</th>
<th>second_message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bye! \0 ...</td>
<td>Hi There \0 ...</td>
</tr>
<tr>
<td>Joe Montana \0</td>
<td>Bye! \0ere \0</td>
</tr>
<tr>
<td>Oh no! \0</td>
<td>Bye! \0! \0</td>
</tr>
<tr>
<td>ByeBye! \0</td>
<td>ByeByeMontana \0</td>
</tr>
</tbody>
</table>

Possible Definitions for strcpy():

Array Notation:

```c
char* strcpy(char* s1, const char* s2) {
    int i = 0;
    while ((s1[i] = s2[i]) != '\0') // or while (s1[i] = s2[i])
        i++;
    return s1;
}
```

Pointer Notation:

```c
char* strcpy(char* s1, const char* s2) {
    char* s = s1;
    while (*s++ = *s2++)
        continue;
    return s1;
}
```
1. **Rectangular array**: pets is an array of 3 subarrays; each consisting of 6 characters

   ```
   char pets[3][6] = { "bird",
   "cat",
   "snake" }; 
   ```

   **Local array on stack**

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>i</td>
<td>r</td>
<td>d</td>
<td>\0</td>
<td>\0</td>
</tr>
<tr>
<td>c</td>
<td>a</td>
<td>t</td>
<td>\0</td>
<td>\0</td>
<td>\0</td>
</tr>
<tr>
<td>s</td>
<td>n</td>
<td>a</td>
<td>k</td>
<td>e</td>
<td>\0</td>
</tr>
</tbody>
</table>

   Recall that the uninitialized elements of a partially initialized local array are automatically set to zero.

2. **Ragged array**: vegies is an array of pointers-to-char

   ```
   char* vegies[3] = { "onion",
   "beet",
   "carrot" }; 
   ```

   **Local array**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>o n i o n</td>
<td>b e e t</td>
<td>c a r r o t</td>
</tr>
</tbody>
</table>

   Recall that a quoted string represents its address.

   ```
   char (* vegies)[3]; // This vegies is a pointer to an array of 3 chars. 
   ```

   **Note:**
   A rectangular array has the potential to waste memory.
// RectVRag1: Rectangular Array versus Ragged Array - Array Notation

#include <iostream> // cout
using namespace std;

int main()
{
    // Rectangular array: array of arrays of chars
    char pets[3][6] = { "bird",
                       "cat",
                       "snake"  };

    int i;
    cout << "Rectangular array, pets:\n";
    for (i = 0; i < 3; i++)
        cout << pets[i] << "\n"; // pets[i] is an array of 6 chars
        // pets[i] contains a string
        // pets[i] == &pets[i][0], the address of the string

    char* vegies[3] = { "onion",
                        "beet",
                        "carrot"  };

    cout << "\nRagged array, vegies:\n";
    for (i = 0; i < 3; i++)
        cout << vegies[i] << "\n"; // vegies[i] is a pointer-to-char
        // vegies[i] contains the address of the 1st char of ith string
        // Each pointer points to the first character
        // of the corresponding string.

    return 0;
}

Output:
Rectangular array, pets:
bird
cat
snake

Ragged array, vegies:
onion
beet
carrot
```
// RectVRag2: Rectangular Array versus Ragged Array - Pointer Notation
#include <iostream>    // cout
using namespace std;

int main()
{
    // Rectangular array: array of arrays of chars
    char pets[3][6] = { "bird",
                        "cat",
                        "snake" };
    cout << "Rectangular array, pets:
";
    char (* pca)[6]; // pointer to array of 6 chars
    for (pca = pets; pca < pets + 3; pca++)
    {
        cout << *pca << "\n"; // pets == &pets[0]
        // pets[0] is an array of 6 chars
        // pca is a pointer to an array of 6 chars
    }

    // Ragged array: array of pointers-to-char
    char* vegies[3] = { "onion",
                        "beet",
                        "carrot" };
    cout << "\nRagged array, vegies:
";
    for (char** ppc = vegies; ppc < vegies + 3; ppc++)
    {
        cout << *ppc << "\n"; // vegies == &vegies[0]
        // vegies[0] contains the address of the 1st char of 1st string
        // vegies is the address of the address of the first string
        // vegies is a pointer to pointer to char
    }
    return 0;
}

Output:
Rectangular array, pets:
bird
cat
snake
Ragged array, vegies:
onion
beet
carrot
```
// RectArrNames: Rectangular array of names

#include <iostream>  // cout, getline()
using namespace std;

const int MAX = 6;  // Maximum number of names
const int SIZE = 81;  // Maximum string length plus 1

int main()
{
    char names[MAX][SIZE];  // Rectangular array: array of arrays of chars

cout << "Enter up to " << MAX << " names.\n" << "To quit, press <Enter> or CTRL+Z\n" << "Enter the first name==>";

int ct = 0;  // names count

while (ct < MAX &&  // Test for maximum names
        cin.getline(names[ct], SIZE) &&  // Get the name
        names[ct][0] != '\0')  // Test for <Enter> key
{
    ct++;
    if (ct < MAX)  // Prompt only if user can enter another name
    cout << "Enter the next name==>";
}

cout << "\nThe names are:\n";
for (int i = 0; i < ct; i++)
    cout << names[i] << '\n';

return 0;
}

Run 1:
Enter up to 6 names.
To quit, press <Enter> or CTRL+Z
Enter the first name==>Henry Gee
Enter the next name==>Edna R. Ebert
Enter the next name==>Chris Burgos
Enter the next name==>Nam Tran
Enter the next name==>Doris Finney
Enter the next name==>Robert Godwin

The names are:
Henry Gee
Edna R. Ebert
Chris Burgos
Nam Tran
Doris Finney
Robert Godwin
Run 2:
Enter up to 6 names.
To quit, press <Enter> or CTRL+Z
Enter the first name==>Chelsea Clinton
Enter the next name==>Mara Grace Munrow
Enter the next name==>Gus Goodman
Enter the next name==>Pat S. Wong
Enter the next name==>
The names are:
Chelsea Clinton
Mara Grace Munrow
Gus Goodman
Pat S. Wong

Run 3:
Enter up to 6 names.
To quit, press <Enter> or CTRL+Z
Enter the first name==>Chelsea Clinton
Enter the next name==>Mara Grace Munrow
Enter the next name==>Gus Goodman
Enter the next name==>Pat S. Wong
Enter the next name==>^Z
The names are:
Chelsea Clinton
Mara Grace Munrow
Gus Goodman
Pat S. Wong

Array image during Run 2:

<table>
<thead>
<tr>
<th>names[0]</th>
<th>Chelsea^Clinton\0</th>
</tr>
</thead>
<tbody>
<tr>
<td>names[1]</td>
<td>Mara^Grace^Munrow\0</td>
</tr>
<tr>
<td>names[2]</td>
<td>^^Gus^Goodman\0</td>
</tr>
<tr>
<td>names[3]</td>
<td>Pat^S.^Wong\0</td>
</tr>
<tr>
<td>names[4]</td>
<td>\0</td>
</tr>
<tr>
<td>names[5]</td>
<td></td>
</tr>
</tbody>
</table>
// RagArrNames: Ragged array of names

#include <iostream>  // cout, cerr, getline()
#include <cstring>   // strlen(), strcpy()
#include <cstdlib>   // exit()

using namespace std;

const int MAX = 6;   // Maximum number of names
const int SIZE = 81; // Maximum string length plus 1

int main()
{
    char temp_name[SIZE];  // Temporary name
    char* pnames[MAX];     // Ragged array

    cout << "Enter up to " << MAX << " names.\n"
    << "To quit, press <Enter>\n"
    << "Enter the first name==>";

    int ct = 0; // names count
    while (ct < MAX && // Test for maximum names
            cin.getline(temp_name, SIZE) && // Get the name
            temp_name[0] != '\0') // Test for Enter key
    {
        pnames[ct] = new char [strlen(temp_name) + 1];
        if (!pnames[ct])
        {
            cerr << "Could not allocate memory";
            exit(1);
        }
        strcpy(pnames[ct], temp_name);
        ct++;
        if (ct < MAX) // Prompt only if user can enter another name
            cout << "Enter the next name==>";
    }

    cout << \n    "\nThe names are:\n";
    for (int i = 0; i < ct; i++)
        cout << pnames[i] << \n';

    for (i = 0; i < ct; i++)
        delete [] pnames[i]; // Deallocate memory holding the names
    return 0;
}
Run 1:
Enter up to 6 names.
To quit, press <Enter>
Enter the first name==>Henry Gee
Enter the next name==>Edna R. Ebert
Enter the next name==>Chris Burgos
Enter the next name==>Nam Tran
Enter the next name==>Doris Finney
Enter the next name==>Robert Godwin

The names are:
Henry Gee
Edna R. Ebert
Chris Burgos
Nam Tran
Doris Finney
Robert Godwin

Run 2:
Enter up to 6 names.
To quit, press <Enter>
Enter the first name==>Chelsea Clinton
Enter the next name==>Mara Grace Munrow
Enter the next name==>Gus Goodman
Enter the next name==>Pat S. Wong

The names are:
Chelsea Clinton
Mara Grace Munrow
Gus Goodman
Pat S. Wong

Ragged array after data entry during Run 2:

```
<table>
<thead>
<tr>
<th>pnames</th>
<th>Free Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Chelsea\Clinton\0</td>
</tr>
<tr>
<td>1</td>
<td>Mara\Grace\Munrow\0</td>
</tr>
<tr>
<td>2</td>
<td>Gus\Goodman\0</td>
</tr>
<tr>
<td>3</td>
<td>Pat\S.Wong\0</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Buffer array for first name:
temp_name  Chelsea\Clinton\0
1. Added with ANSI/ISO standard

2. English mathematician, George Boole, who developed a mathematical representation of the laws of logic.

Boolean Algebra

3. Boolean variable has one of the two values: true or false

4. `bool received = true;`

   `bool` constants: true and false

5. `bool` true and false can be promoted to `int` with true converting to 1 and false to 0

   ```
   int yes = true;     // yes initialized to 1
   int no = false;     // no initialized to 0
   ```

6. Numeric values can be converted to `bool`

   ```
   bool yea = 100;     // yea initialized to true
   bool nay = 0;       // nay initialized to false
   ```

7. `bool odd(int num)` // Returns true if argument is odd

   ```
   return num % 2;  // or return (num % 2 ? true : false);
   }
   ```

   `bool even(int num)` // Returns true if argument is even

   ```
   return !(num % 2);  // or return (num % 2 ? false : true);
   ```
8. `cout << odd(n);` // bool type is displayed as 0 or 1
   `cout << boolalpha << odd(n);` // displayed as false or true
   // manipulator converts bool to
   // "false" or "true" strings

9. `bool b;`
   `cin >> b;` // 0 sets b to false; non-zero sets b to true

10. `sizeof (bool)` is 1