

Arrays, Pointers and Functions

2.1 Arrays

Arrays in C++ are declared to hold a specific number of the same type of object. The valid indices are 0 up to 1 less than the size of the array. The execution does no checking for references going outside the bounds of the array. Arrays can be initialized at declaration.

2.2 Functions

A function is a self-standing piece of code. It can return a variable of a specified type, or have type `void`. It can have arguments of specific type. In general, variables are passed *by value*, which means that the function receives a copy of the variable. This is inefficient for large objects, so these are usually passed *by address* (such as automatically occurs for arrays) or *by reference* (discussed later). To aid the compiler, a *prototype* of a function is used at the start of a program to tell the compiler of the existence of such a function: the prototype specifies the name, arguments, and type of the function.

2.3 Pointers

A *pointer* stores an address. A pointer has a type: this indicates the type of object stored at the address to which the pointer points. A pointer is defined using the `*`, and is dereferenced thereby too. An array name is equivalent to a pointer to the start of that array. Primitive arithmetic can be applied to pointers.

2.4 C-Strings

There are two options to store strings in C++. The first is the way done in C, now called C-strings; the second is with the object `string`, discussed later. A C-string is stored in an array of `chars`, terminated by the null character, denoted `'\0'` or simply `0` itself. The user is responsible for ensuring that the null terminator remains present. Constant strings defined by the user using quotation marks are automatically C-strings. With the `cstring` library, strings can be compared, they can be cin-ed and cout-ed, they can be copied, appended, and several other things. C-strings are passed to functions by reference: that is, by supplying the address of the first character using the array name or a char pointer.

2.5 Sample program: primality.cpp

```
// print out primes less than 100
// wdg 2009
#include <iostream>
using namespace std;
```

```
bool isPrime(int x);
const int MAX=100;
```

```
int main( )
{
    int test=2;
    while( test<MAX ) {
        if( isPrime(test) )
        cout << test << " ";
        test++;
    }
    cout << endl;
    return 0;
}
```

```
bool isPrime(int x)
{
    for(int y=2; y<x; y++)
        if( x%y==0 )
            return false;
    return true;
}
```

2.6 Sample program: strstr.cpp

```
// strstr.cpp - adapted from nkraft by wdg 2009
// Find first occurrence of substring needle in string haystack.
#include <iostream>
using namespace std;

char *strstr ( char *haystack, char *needle );
```

```

int main ( )
{
    char *one = "concatenate";
    char *two = "cat";
    char *three = "dog";

    char *ans = strstr( one, two );
    if ( !ans )
        cout << "needle1 not found\n";
    else
        cout << "needle1 starts at index " << ans-one
                << " of haystack" << endl;
    ans = strstr( one, three );
    if ( !ans )
        cout << "needle2 not found\n";
    else
        cout << "needle2 starts at index " << ans-one
                << " of haystack" << endl;

    return 0;
}

// The function returns pointer to beginning of substring,
// or NULL if substring is not found.
char *strstr ( char *haystack, char *needle )
{
    char *start;
    for(start = haystack; *start != '\0'; start++ ) {
        char *p = needle;
        char *q = start;
        while ( *p != '\0' && *q != '\0' && *p == *q ) {
            p++;
            q++;
        }
        if( *p == '\0' )
            return start;    // reached end of needle without mismatch
    }
    return NULL;
}

```