Standard C-library Input Functions

The C function library is a part of the standard C++ library. Rather than try to introduce a specific class library at this point, we will use the standard C function library. For the exam, it will be assumed that you know the functions in the standard C library so you should take the opportunity to practise. The informal coursework is designed to help you to practise these, as well as to better understand pointers.

1 Useful C-library input functions

You should refer to the lecture slides and samples for Lecture 5 when considering these functions. The following three functions are useful for reading input that is typed by a user:

int getchar();

Read a single character. See, for example: `http://www.cplusplus.com/reference/clibrary/cstdio/getchar.html`

char* gets (char* str);

Read an entire string, up to the new-line character. Warning: There is no checking of string length. See, for example: `http://www.cplusplus.com/reference/clibrary/cstdio/gets.html`

scanf()

This function has a lot of power and flexibility. I strongly suggest that you read the documentation about both `printf()` and `scanf()` and ensure that you understand the power of these functions.

You should also try the sample code from section 4, which will show you how you can limit the size of the input that is read to prevent reading more characters than you can store.

1.1 Treating standard input as a file stream

You can also use any of the file functions to read from the standard input stream, in the same way as the above functions. It is useful to know that any file input function which takes a parameter of type `FILE*` can be used with the standard input stream (i.e. the keyboard/command line in a shell, unless input was redirected) by passing the value `stdin` as the `FILE*` parameter. (This value is defined in `cstdio` so you will usually need to `#include` that file.)

This means that you can use the following code to read from the input stream:

int fgetc(FILE* stream);

Read a single character from an input stream, including `stdin`. Where the `stdin` stream is used, this will read from the standard input, i.e. the keyboard (unless input was redirected).

e.g. `int nextchar = fgetc( stdin );`

This is the same as using `getchar()`.

char* fgets(char* str, int num, FILE* stream);

Read a string up to the new line or the maximum specified number of characters, whichever comes first. Where the `stdin` stream is used this will read from the standard input, i.e. the keyboard (unless input was redirected). e.g.

```
char buffer[100];
```

```
if ( fgets( buffer, 100, stdin ) != NULL )
  ... do something with buffer ...
```

This is actually better than the `gets()` function, since it allows you to limit the amount of input which can be received.

2 Examples of using different input functions

```
#include <cstdio>

char* MyGetLine( char* dest, int iMaxDigits )
{
  /* Read in data */
  int iCharNum = 0;

  /* Loop getting each character in turn */
  for ( iCharNum=0 ; iCharNum < (iMaxDigits-1) ; iCharNum++ )
  {
    dest[iCharNum] = getchar();
    if ( dest[iCharNum] == '\n' )
      break;
    if ( dest[iCharNum] == EOF )
      return NULL;
  }

  /* Unlike fgets we DO NOT bother keeping the \n */
  if ( dest[iCharNum] != '\n' )
    while( getchar() != '\n' )
      ; /* skip any chars to the end of line */

  /* Terminate the string that was read */
  dest[iCharNum] = 0;
  return dest;
}

int main( int argc, char* argv[] )
{
  char buffer[128];
  int iChar;

  printf( "Enter some text for getchar() :" );
  iChar = getchar();
  printf( "Character read was %c\n", iChar );
  /* Throw away the rest of the characters up to a newline */
  if ( iChar != '\n' ) while( getchar() != '\n' ) ;

  printf( "\n-\n\n" );

  printf( "Warning: If you provide too much input then the buffer overwrite " );
  printf( "may crash the program at some point later on. Including when " );
  printf( "main() ends!\n\n" );
  printf( "Enter some text for gets() (Warning: not over 127 chars!) : " );
  if ( gets( buffer ) != NULL )
    printf( "String read was %s\n", buffer );

  printf( "\n-\n\n" );

  printf( "Enter some text for fgets() : " );
  if ( fgets( buffer, 128, stdin ) != NULL )
    printf( "String read was %s\n", buffer );
  /* Throw away the rest of the characters up to a newline if newline was not read */
```

```
  if ( buffer[strlen(buffer)-1] != '\n' ) while( getchar() != '\n' ) ;

  printf( "\n-\n\n" );

  printf( "Enter some text for MyGetLine() : " );
  if ( MyGetLine( buffer, 128 ) != NULL )
    printf( "String read was %s\n", buffer );

  return 0;
}
```

2.1 Your own function

You can build your own function to read a string which will limit the number of characters read using `getchar()`. For example, the following code will read a string with a maximum number of characters, then throw away any further characters up to the newline.

```
char* MyGetLine( char* dest, int iMaxDigits )
{
  /* Read in data */
  int iCharNum = 0;
  /* Loop getting each character in turn */
  for ( iCharNum=0 ; iCharNum < (iMaxDigits-1) ; iCharNum++ )
  {
    dest[iCharNum] = getchar();
    if ( dest[iCharNum] == '\n' )
      break;
  }

  /* Unlike fgets(stdin) we DO NOT bother keeping the terminal \n */
  if ( dest[iCharNum] != '\n' )
    while( getchar() != '\n' )
      ; /* skip any chars to the end of line */

  /* Terminate the string that was read */
  dest[iCharNum] = 0;
  return dest;
}
```

3 Examples of using printf() and sizeof()

```
#include <cstdio>

char* strings[] = { "Hello", "Longer string", "A", "A very long string" };

int main( int argc, char* argv[] )
{
  int i;

  printf( "\nPrint a min field size of 2:\n" );
  for ( i = 0 ; i < (sizeof(strings)/sizeof(char*)) ; i++ )
    printf( "%2s\n", strings[i] );

  printf( "\nPrint a min field size of 8:\n" );
  for ( i = 0 ; i < (sizeof(strings)/sizeof(char*)) ; i++ )
    printf( "%8s\n", strings[i] );

  printf( "\nPrint a max field size of 2:\n" );
  for ( i = 0 ; i < (sizeof(strings)/sizeof(char*)) ; i++ )
    printf( "%.2s\n", strings[i] );

  printf( "\nPrint a max field size of 8:\n" );
  for ( i = 0 ; i < (sizeof(strings)/sizeof(char*)) ; i++ )
    printf( "%.8s\n", strings[i] );

  printf( "\nPrint a field of size exactly 4:\n" );
  for ( i = 0 ; i < (sizeof(strings)/sizeof(char*)) ; i++ )
    printf( "%4.4s\n", strings[i] );

  return 0;
}
```

4 Example of using scanf

```
#include <cstdio>

int main( int argc, char* argv[] )
{
  char buffer[16];
  int i;

  printf( "\n\nRead exactly 15 chars (including newlines!):\n" );
  scanf( "%15c", buffer );
  buffer[15] = 0; /* The %c will read the chars but not terminate the string. */
  printf( "Read '%s'\n", buffer );
  if ( buffer[strlen(buffer)-1] != '\n') while( getchar() != '\n' ) ;

  printf( "\n\nRead up to 5 chars, or up to newline:\n" );
  scanf( "%5s", buffer );
  printf( "Read '%s'\n", buffer );
  if ( buffer[strlen(buffer)-1] != '\n') while( getchar() != '\n' ) ;

  printf( "\n\nRead at most 10 chars, or up to newline\n" );
  scanf( "%10s", buffer );
  printf( "Read '%s'\n", buffer );
  if ( buffer[strlen(buffer)-1] != '\n') while( getchar() != '\n' ) ;

  printf( "\n\nRead at most 6 chars, or up to newline\n" );
  scanf( "%6s", buffer );
  printf( "Read '%s'\n", buffer );
  if ( buffer[strlen(buffer)-1] != '\n') while( getchar() != '\n' ) ;

  return 0;
}
```