# *Windows Headers for Strings*

Windows header file types for strings

Windows has two types of strings: 8-bit and 16-bit. 8-bit strings are used for older character sets, such as ASCII.

16-bit strings are used for Unicode, which is a newer character set that can represent characters from all major languages.

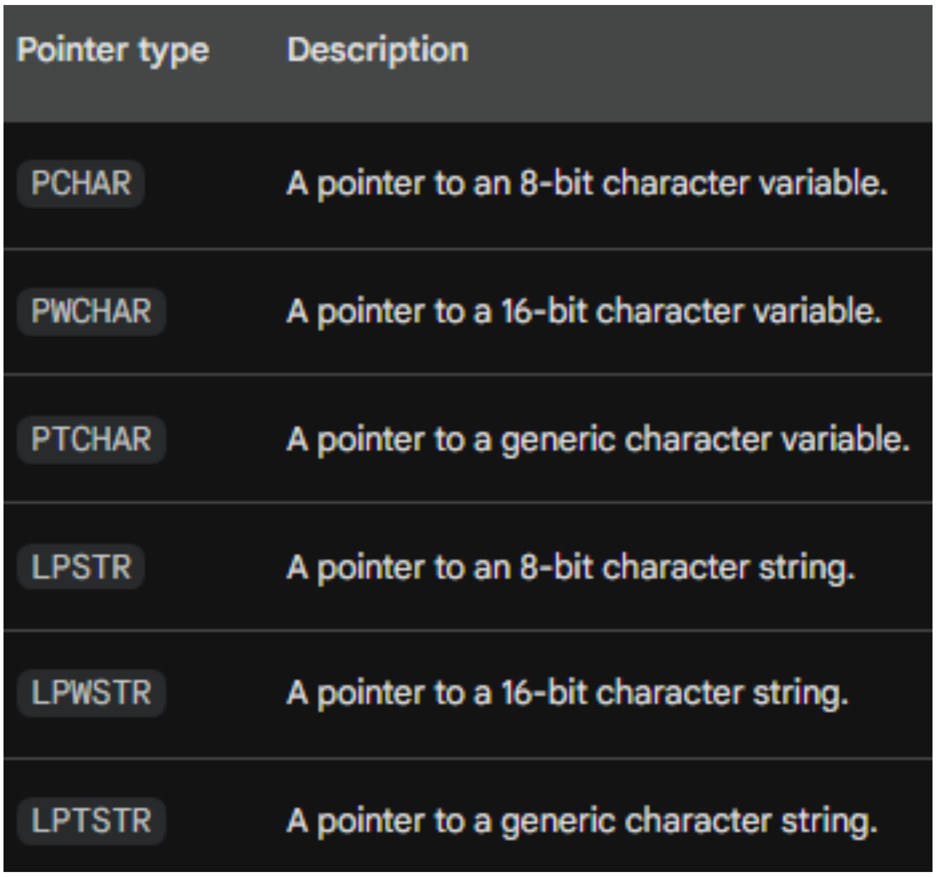
Windows provides a number of header files that define data types for strings. The most important header file is **winnt.h**. This header file defines the following data types:

• **CHAR:** An 8-bit character.

• **WCHAR:** A 16-bit character.

• **TCHAR:** A generic character type that can be either an 8-bit character or a 16-bit character, depending on the definition of the UNICODE identifier.

Windows also provides a number of **pointer types for strings**. These pointer types are used to point to character variables and strings. The following table lists the most important pointer types for strings:



| Pointer type | Description |
|---|---|
| PCHAR | A pointer to an 8-bit character variable. |
| PWCHAR | A pointer to a 16-bit character variable. |
| PTCHAR | A pointer to a generic character variable. |
| LPSTR | A pointer to an 8-bit character string. |
| LPWSTR | A pointer to a 16-bit character string. |
| LPTSTR | A pointer to a generic character string. |

The **N and L prefixes** stand for **"near"** and **"long"** and refer to the two different sizes of pointers in 16-bit Windows. There is no differentiation between near and long pointers in Win32.

## WHICH DATATYPE AND POINTER SHOULD I USE?

If you are writing a program that will only be used with older character sets, such as ASCII, you can use the CHAR data type and the LPSTR pointer type.

If you are writing a program that will be used with Unicode, you should use the WCHAR data type and the LPWSTR pointer type.

If you are writing a program that will be used with both older character sets and Unicode, you should use the TCHAR data type and the LPTSTR pointer type.

## HOW TO DEFINE STRINGS

To define an 8-bit character string, you can use the following syntax:

```c
char *string = "Hello, world!";
```

To define a 16-bit character string, you can use the following syntax:

```c
wchar_t *string = L"Hello, world!";
```

To define a generic character string, you can use the following syntax:

```c
TCHAR *string = TEXT("Hello, world!");
```

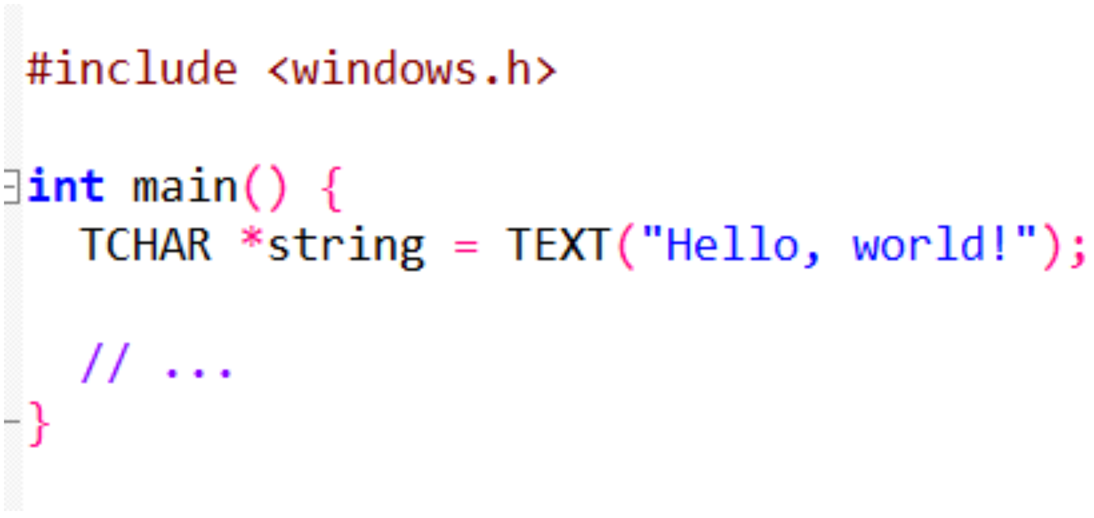
When I say that the **TCHAR data type is generic**, I mean that it can be either Unicode formatted or ASCII, depending on the definition of the UNICODE identifier.

If the **UNICODE identifier is defined**, then the TCHAR data type is defined to be the same as the WCHAR data type. This means that TCHAR variables and strings will be Unicode formatted.

If the **UNICODE identifier is not defined**, then the TCHAR data type is defined to be the same as the CHAR data type. This means that TCHAR variables and strings will be ASCII formatted.

This allows you to write programs that can be compiled for either Unicode or ASCII, simply by defining or not defining the UNICODE identifier.

For example, the following code will compile for either Unicode or ASCII:



```
#include <windows.h>

int main() {
   TCHAR *string = TEXT("Hello, world!");

   // ...
}
```

If you **compile this code with the UNICODE identifier defined**, then the string variable will be a Unicode string.

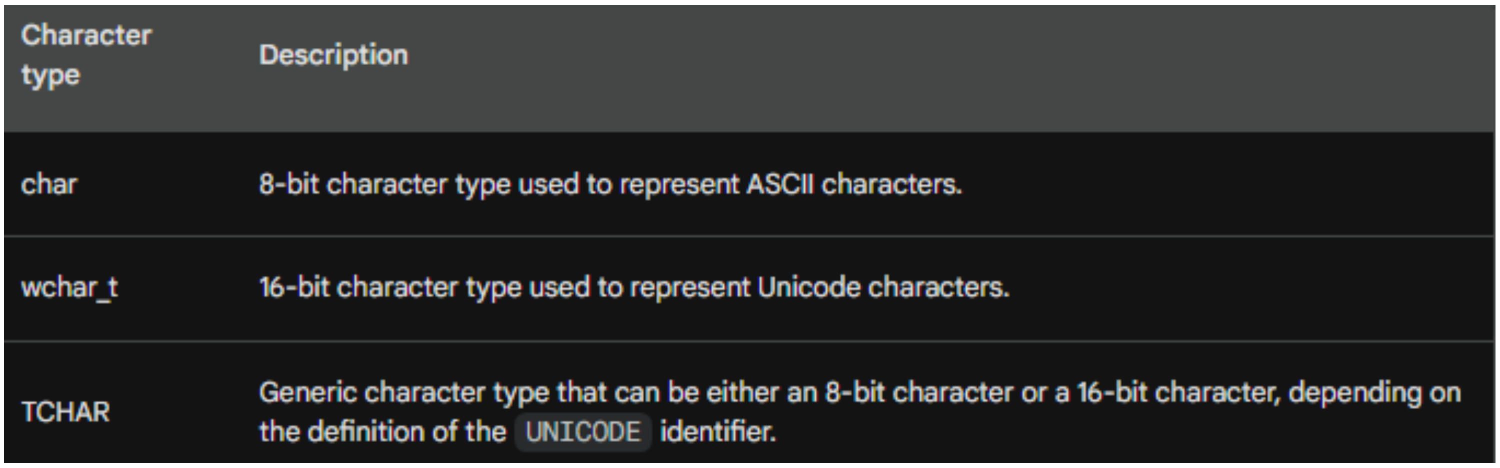
If you **compile this code without the UNICODE identifier defined**, then the string variable will be an ASCII string.

This makes it easy to write programs that are compatible with both older character sets and Unicode.

*======================================*

***So, char above == ASCII, wchar == UNICODE, TCHAR == varies?***

Yes,



| Character type | Description |
|---|---|
| char | 8-bit character type used to represent ASCII characters. |
| wchar_t | 16-bit character type used to represent Unicode characters. |
| TCHAR | Generic character type that can be either an 8-bit character or a 16-bit character, depending on the definition of the `UNICODE` identifier. |

If the UNICODE identifier is defined, then TCHAR is the same as wchar_t.

If the UNICODE identifier is not defined, then TCHAR is the same as char.

When to use each character type:

- **Use char if** you are writing a program that will only be used with older character sets, such as ASCII.

- **Use wchar_t if** you are writing a program that will be used with Unicode.

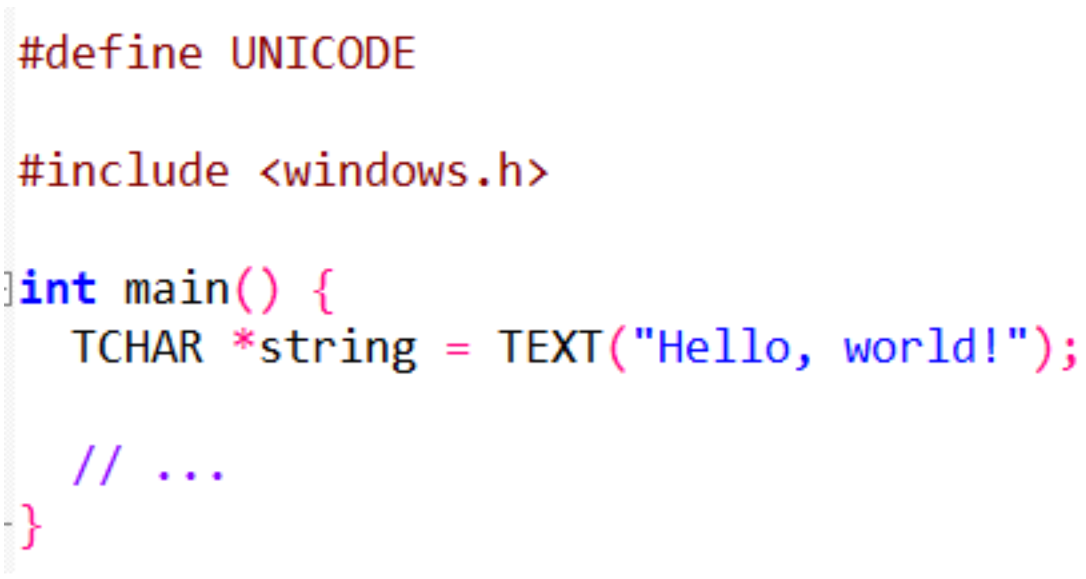
- **Use TCHAR if** you are writing a program that will be used with both older character sets and Unicode.

By using the appropriate character type, you can write programs that are compatible with a wide range of character sets and operating systems.

=====================================

***So, how will i define that unicode identifier or how will i leave it undefined?***

This will compile to a Unicode program:



```
#define UNICODE

#include <windows.h>

int main() {
  TCHAR *string = TEXT("Hello, world!");

  // ...
}
```

This will compile to an ASCII:

```c
#include <windows.h>

int main() {
  TCHAR *string = TEXT("Hello, world!");

  // ...
}
```

You can also define the UNICODE identifier at the command line when you compile your program.

To do this, you use the **/DUNICODE compiler option.**

For example, the following command will compile the above code to a Unicode program:

```
cl /DUNICODE main.c
```

To leave the UNICODE identifier undefined, you simply **omit the / DUNICODE compiler option.**

## WHICH OPTION SHOULD YOU USE?

If you are writing a program that will only be used with older character sets, such as ASCII, then you should leave the UNICODE identifier undefined.

If you are writing a program that will be used with Unicode, then you should define the UNICODE identifier.

If you are writing a program that will be used with both older character sets and Unicode, then you should use the TCHAR data type and the TEXT macro. This will allow your program to be compiled for either Unicode or ASCII, depending on how you compile it.

***Conclusion***

By understanding how to define and leave undefined the UNICODE identifier, you can write programs that are compatible with both older character sets and Unicode.

=============================================

***Why did we define Unicode as a constant on top of our program, but without any data eg***

```
#define AGE 12
#define PERCENTAGE 100
#define TEETH 32
#define PI 3.1459
```

You do not need to define Unicode as a constant on top of your program, and it is not recommended to do so.

The Unicode identifier is a **preprocessor macro that is defined in the winnt.h header file.** It is used to tell the compiler whether to compile the program for Unicode or ASCII.

If you define Unicode as a constant on top of your program, you are overriding the definition in the winnt.h header file. This can lead to unexpected results, and it is not recommended.

Instead of defining Unicode as a constant, you should simply include the winnt.h header file at the top of your program. This will ensure that the Unicode identifier is defined correctly.

For example, the following code is correct:

```
#include <winnt.h>

int main() {
  TCHAR *string = TEXT("Hello, world!");

  // ...
}
```

The winnt.h header file must be included at the top of any program that uses the TCHAR data type or the TEXT macro.

This is because the **winnt.h header file defines the Unicode identifier,** which tells the compiler whether to compile the program for Unicode or ASCII.

I apologize for my previous mistake.