2.3 Identifiers

Rules for identifiers

A name created by a programmer for an item like a variable or function is called an ***identifier***. An identifier must:

- be a sequence of letters (a-z, A-Z), underscores (_), and digits (0-9)
- start with a letter or underscore

Note that "_", called an underscore, is considered to be a letter.

Identifiers are ***case sensitive***, meaning upper and lower case letters differ. So numCats and NumCats are different.

A ***reserved word*** is a word that is part of the language, like int, short, or double. A reserved word is also known as a ***keyword***. A programmer cannot use a reserved word as an identifier. Many language editors will automatically color a program's reserved words. A list of reserved words appears at the end of this section.

PARTICIPATION ACTIVITY 2.3.1: Identifier validator.

Check if the following identifiers are valid: c, cat, n1m1, short1, _hello, 42c, hi there, and cat! (Note: Doesn't consider library items.)

Try an identifier: Validate

Awaiting your input...

Feedback?

PARTICIPATION ACTIVITY 2.3.2: Valid identifiers.

Which are valid identifiers?

1) numCars

- (•) Valid
- ( ) Invalid

Correct

Letters are always valid (upper or lower case).

2) num_Cars1

- (•) Valid
- ( ) Invalid

Correct

An underscore is treated as a letter.

3) _numCars

- (•) Valid
- ( ) Invalid

Correct

The underscore can be in the first position just like any other letter.

4) ___numCars

- (•) Valid
- ( ) Invalid

Correct

Two or more underscores may look strange, but the underscore is just like any other letter and can be repeated. However, convention is to not use more than one underscore.

5) 3rdPlace

- ( ) Valid
- (•) Invalid

Correct

Identifiers must start with a letter.

6) thirdPlace_

- (•) Valid
- ( ) Invalid

Correct

Ending with an underscore is okay.

7) thirdPlace!

- ( ) Valid
- (•) Invalid

Correct

"!" is not allowed.

8) short

- ( ) Valid
- (•) Invalid

Correct

"short" is a language keyword so it can't be used as an identifier.

9) very tall

- ( ) Valid
- (•) Invalid

Correct

Spaces are not allowed. Instead, an underscore can be used (very_tall) or the words can be abutted (veryTall).

Feedback?

Style guidelines for identifiers

While various (crazy-looking) identifiers may be valid, programmers may follow identifier naming conventions (style) defined by their company, team, teacher, etc. Two common conventions for naming variables are:

- Camel case: ***Lower camel case*** abuts multiple words, capitalizing each word except the first, as in numApples or peopleOnBus.
- Underscore separated: Words are lowercase and separated by an underscore, as in num_apples or people_on_bus.

Neither convention is better. The key is to be consistent so code is easier to read and maintain.

Good practice is to create meaningful identifier names that self-describe an item's purpose. Good practice minimizes use of abbreviations in identifiers except for well-known ones like num in numPassengers. Programmers must strive to find a balance. Abbreviations make programs harder to read and can lead to confusion. Long variable names, such as averageAgeOfUclaGraduateStudent may be meaningful, but can make subsequent statements too long and thus hard to read.

PARTICIPATION ACTIVITY 2.3.3: Meaningful identifiers.

Choose the "best" identifier for a variable with the stated purpose, given the above discussion.

1) The number of students attending UCLA.
   - num
   - numStdsUcla
   - numStudentsUcla
   - numberOfStudentsAttendingUcla

Correct

Ucla is likely a well-known abbreviation, so shouldn't cause a problem.

2) The size of an LCD monitor
   - size
   - sizeLcdMonitor
   - s
   - sizeLcdMtr

Correct

Lcd is abbreviated but may be OK.

3) The number of jelly beans in a jar.
   - numberOfJellyBeansInTheJar
   - jellyBeansInJar
   - nmJlyBnsInJr

Correct

Looks OK. Could start with num, but should be fine as is.

Feedback?

zyBook's naming conventions

Lower camel case is used for variable naming. This material strives to follow another good practice of using two or more words per variable such as numStudents rather than just students, to provide meaningfulness, to make variables more recognizable when variable names appear in writing like in this text or in a comment, and to reduce conflicts with reserved words or other already-defined identifiers.

Table 2.3.1: C++ reserved words / keywords.

alignas (since C++11)	decltype (since C++11)	namespace	struct
alignof (since C++11)	default	new	switch
and	delete	noexcept (since C++11)	template
and_eq	do	not	this
asm	double	not_eq	thread_local (since C++11)
auto	dynamic_cast	nullptr (since C++11)	throw
bitand	else	operator	true
bitor	enum	or	try
bool	explicit	or_eq	typedef
break	export	private	typeid
case	extern	protected	typename
catch	false	public	union
char	float	register	unsigned
char16_t (since C++11)	for	reinterpret_cast	using
char32_t (since C++11)	friend	return	virtual
class	goto	short	void
compl	if	signed	volatile
const	inline	sizeof	wchar_t
constexpr (since C++11)	int	static	while
const_cast	long	static_assert (since C++11)	xor
continue	mutable	static_cast	xor_eq

Source: http://en.cppreference.com/w/cpp/keyword.

Feedback?