WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Čeština
Ελληνικά
Esperanto
Français
Bahasa Indonesia
Italiano
Português
Русский
Suomi
Українська
Edit links

Record (computer science)

From Wikipedia, the free encyclopedia

In computer science, a **record** (also called **struct** or **compound data**)[1][*page needed*] is a basic data structure (a tuple may or may not be considered a record, and vice versa, depending on conventions and the programming language at hand). A record is a collection of *elements*, typically in fixed number and sequence and typically indexed by serial numbers or identity numbers. The elements of records may also be called *fields* or *members*.

For example, a date could be stored as a record containing a numeric *year* field, a *month* field represented as a string, and a numeric *day-of-month* field. As another example, a Personnel record might contain a *name*, a *salary*, and a *rank*. As yet another example, a Circle record might contain a *center* and a *radius*. In this instance, the center itself might be represented as a Point record containing *x* and *y* coordinates.

Records are distinguished from arrays by the fact that their number of fields is typically fixed, each field has a name, and that each field may have a different type.

A **record type** is a data type that describes such values and variables. Most modern computer languages allow the programmer to define new record types. The definition includes specifying the data type of each field and an identifier (name or label) by which it can be accessed. In type theory, product types (with no field names) are generally preferred due to their simplicity, but proper record types are studied in languages such as System F-sub. Since type-theoretical records may contain first-class function-typed fields in addition to data, they can express many features of object-oriented programming.

Records can exist in any storage medium, including main memory and mass storage devices such as magnetic tapes or hard disks. Records are a fundamental component of most data structures, especially linked data structures. Many computer files are organized as arrays of logical records, often grouped into larger physical records or blocks for efficiency.

The parameters of a function or procedure can often be viewed as the fields of a record variable; and the arguments passed to that function can be viewed as a record value that gets assigned to that variable at the time of the call. Also, in the call stack that is often used to implement procedure calls, each entry is an *activation record* or *call frame*, containing the procedure parameters and local variables, the return address, and other internal fields.

An object in object-oriented language is essentially a record that contains procedures specialized to handle that record; and object types are an elaboration of record types. Indeed, in most object-oriented languages, records are just special cases of objects, and are known as plain old data structures (PODSs), to contrast with objects that use OO features.

A record can be viewed as the computer analog of a mathematical tuple. In the same vein, a record type can be viewed as the computer language analog of the Cartesian product of two or more mathematical sets, or the implementation of an abstract product type in a specific language.

Contents [hide]

History [edit]

The concept of record can be traced to various types of tables and ledgers used in accounting since remote times. The modern notion of records in computer science, with fields of well-defined type and size, was already implicit in 19th century mechanical calculators, such as Babbage's Analytical Engine.[*citation needed*]

Records were well established in the first half of the 20th century, when most data processing was done using punched cards. Typically each record of a data file would be recorded in one punched card, with specific columns assigned to specific fields. Generally, a record was the smallest unit that could be read in from external storage (e.g. card reader, tape or disk).

Most machine language implementations and early assembly languages did not have special syntax for records, but the concept was available (and extensively used) through the use of index registers, indirect addressing, and self-modifying code. Some early computers, such as the IBM 1620, had hardware support for delimiting records and fields, and special instructions for copying such records.

The concept of records and fields was central in some early file sorting and tabulating utilities, such as IBM's Report Program Generator (RPG).

COBOL was the first widespread programming language to support record types,[2] and its record definition facilities were quite sophisticated at the time. The language allows for the definition of nested records with alphanumeric, integer, and fractional fields of arbitrary size and precision, as well as fields that automatically format any value assigned to them (e.g., insertion of currency signs, decimal points, and digit group separators). Each file is associated with a record variable where data is read into or written from. COBOL also provides a `MOVE` `CORRESPONDING` statement that assigns corresponding fields of two records according to their names.

The early languages developed for numeric computing, such as FORTRAN (up to FORTRAN IV) and Algol 60, did not have support for record types; but latter versions of those languages, such as Fortran 77 and Algol 68 did add them. The original Lisp programming language too was lacking records (except for the built-in cons cell), but its S-expressions provided an adequate surrogate. The Pascal programming language was one of the first languages to fully integrate record types with other basic types into a logically consistent type system. IBM's PL/1 programming language provided for COBOL-style records. The C programming language initially provided the record concept as a kind of template (`struct`) that could be laid on top of a memory area, rather than a true record data type. The latter were provided eventually (by the `typedef` declaration), but the two concepts are still distinct in the language. Most languages designed after Pascal (such as Ada, Modula, and Java) also supported records.

Operations [edit]

A programming language that supports record types usually provides some or all of the following operations:

- Declaration of a new record type, including the position, type, and (possibly) name of each field;
- Declaration of variables and values as having a given record type;
- Construction of a record value from given field values and (sometimes) with given field names;
- Selection of a field of a record with an explicit name;
- Assignment of a record value to a record variable;
- Comparison of two records for equality;
- Computation of a standard hash value for the record.

The selection of a field from a record value yields a value.

Some languages may provide facilities that enumerate all fields of a record, or at least the fields that are references. This facility is needed to implement certain services such as debuggers, garbage collectors, and serialization. It requires some degree of type polymorphism.

In systems with record subtyping, operations on values of record type may also include:

- Adding a new field to a record, setting the value of the new field.
- Removing a field from a record.

In such settings, a specific record type implies that a specific set of fields are present, but values of that type may contain additional fields. A record with fields *x*, *y*, and *z* would thus belong to the type of records with fields *x* and *y*, as would a record with fields *x*, *y*, and *r*. The rationale is that passing an (*x*,*y*,*z*) record to a function that expects an (*x*,*y*) record as argument should work, since that function will find all the fields it requires within the record. Many ways of practically implementing records in programming languages would have trouble with allowing such variability, but the matter is a central characteristic of record types in more theoretical contexts.

Assignment and comparison [edit]

Most languages allow assignment between records that have exactly the same record type (including same field types and names, in the same order). Depending on the language, however, two record data types defined separately may be regarded as distinct types even if they have exactly the same fields.

Some languages may also allow assignment between records whose fields have different names, matching each field value with the corresponding field variable by their positions within the record; so that, for example, a complex number with fields called `real` and `imag` can be assigned to a 2D point record variable with fields `X` and `Y`. In this alternative, the two operands are still required to have the same sequence of field types. Some languages may also require that corresponding types have the same size and encoding as well, so that the whole record can be assigned as an uninterpreted bit string. Other languages may be more flexible in this regard, and require only that each value field can be legally assigned to the corresponding variable field; so that, for example, a short integer field can be assigned to a long integer field, or vice versa.

Other languages (such as COBOL) may match fields and values by their names, rather than positions.

These same possibilities apply to the comparison of two record values for equality. Some languages may also allow order comparisons ('<'and '>'), using the lexicographic order based on the comparison of individual fields.[*citation needed*]

PL/I allows both of the preceding types of assignment, and also allows *structure expressions*, such as `a = a+1;` where "a" is a record, or structure in PL/I terminology.

Algol 68's distributive field selection [edit]

In Algol 68, if `Pts` was an array of records, each with integer fields `X` and `Y`, one could write `Pts.Y` to obtain an array of integers, consisting of the `Y` fields of all the elements of `Pts`. As a result, the statements `Pts[3].Y := 7` and `Pts.Y[3] := 7` would have the same effect.

Pascal's "with" statement [edit]

In the Pascal programming language, the command `with R do S` would execute the command sequence `S` as if all the fields of record `R` had been declared as variables. So, instead of writing `Pt.X := 5; Pt.Y := Pt.X + 3` one could write `with Pt do begin X := 5; Y := X + 3 end`.

Representation in memory [edit]

The representation of records in memory varies depending on the programming languages. Usually the fields are stored in consecutive positions in memory, in the same order as they are declared in the record type. This may result in two or more fields stored into the same word of memory; indeed, this feature is often used in systems programming to access specific bits of a word. On the other hand, most compilers will add padding fields, mostly invisible to the programmer, in order to comply with alignment constraints imposed by the machine —say, that a floating point field must occupy a single word.

Some languages may implement a record as an array of addresses pointing to the fields (and, possibly, to their names and/or types). Objects in object-oriented languages are often implemented in rather complicated ways, especially in languages that allow multiple class inheritance.

Examples [edit]

The following show examples of record definitions:

- PL/I:

```
declare 1 date,
          2 year  picture '9999',
          2 month picture '99',
          2 day   picture '99';
```

- C:

```
struct date {
   int year;
   int month;
   int day;
};
```

See also [edit]

- Block (data storage)
- Composite data type
- Cons cell
- Data hierarchy
- Data structure alignment
- Object composition
- Row (database)
- struct (C programming language)
- Storage record

References [edit]

1. ^ Felleisen et al., *How To Design Programs*, MIT Press, 2001
2. ^ Sebesta, Robert W. *Concepts of Programming Languages* (Third ed.). Addison-Wesley Publishing Company, Inc. p. 218. ISBN 0-8053-7133-8.

v · t · e	**Data types** [hide]
Uninterpreted	Bit · Byte · Trit · Tryte · Word
Numeric	Bignum · Complex · Decimal · Fixed point · Floating point (Double precision · Extended precision · Half precision · Minifloat · Octuple precision · Quadruple precision · Single precision) · Integer (signedness) · Interval · Rational
Text	Character · String (null-terminated)
Pointer	Address (physical · virtual) · Reference
Composite	Algebraic data type (generalized) · Array · Associative array · Class · Dependent · Equality · Inductive · List · Object (metaobject) · Option type · Product · **Record** · Set · Union (tagged)
Other	Boolean · Bottom type · Collection · Enumerated type · Exception · Function type · Opaque data type · Recursive data type · Semaphore · Stream · Top type · Type class · Unit type · Void
Related topics	Abstract data type · Data structure · Generic · Kind (metaclass) · Parametric polymorphism · Primitive data type · Protocol (interface) · Subtyping · Type constructor · Type conversion · Type system

Categories: Data types | Composite data types