C

De standaard programmeertaal

Oorsprong

CPL stond voor Combined Programming Language of Cambridge Programming Language. Ze stamt uit 1963, maar de eerste compiler arriveerde pas rond 1970. De taal was gebaseerd op Algol-60 en zou low-level en high-level programmeren combineren.

Omdat de taal nogal complex was verscheen in 1966 BCPL (Basic Combined Programming Language), een eenvoudige gestructureerde imperatieve taal zonder data types, dat vooral bedoeld was om compilers in te schrijven en er verschenen compilers voor tal van computers.

In 1969 ontwikkelden Ken Thompson en Dennis Ritchie de B programmeertaal door overbodige elementen uit BCPL te verwijderen. De syntax werd echter flink omgegooid en een stuk beknopter gemaakt, zodat ze veel op C leek. B en BCPL zijn praktisch verdwenen.

De eerste compiler draaide onder Unix op de PDP-7 en PDP-11. De taal werd opgevolgd door New B (NB) en C. C werd geschreven door Dennis Ritchie en Ken Thompson en de eerste compiler verscheen in 1972 op de PDP-11. In 1973 was de Unix kernel grotendeels in C geschreven. C en Unix bleken erg portabel te zijn en draaien tegenwoordig op vrijwel alle computers.

In 1989 bracht de ANSI C standaard een flinke verbetering, vooral in compatibiliteit. Uitgebreidere standaarden verschenen in 1999 en 2011.

Tegenwoordig is C de taal waarin de meest gebruikte en belangrijkste programma's geschreven zijn. Vooral in systeemprogrammering, embedded computers en supercomputers is ze dominant.

Toen object-georiënteerd programmeren in de mode kwam, schreef Bjarne Stroustrup een uitbreiding die C++ heette, terwijl Objective C vooral bij Apple is gebruikt. Java van Brian Gosling is een sterk versimpeld C++, terwijl C# van Microsoft Corporation een soort Java voor Windows is. Invloeden van de syntax van C vind je in een hoop talen zoals Python.

De D programmeertaal werd in 2001 gepubliceerd door Walter Bright als opvolger van C++ en Java, het is object-georiënteerd met functionele features en ook geschikt voor low-level programmeren. De laatste tijd wordt Rust aangeprezen als opvolger van C.

Kenmerken van de taal

C heeft enkele eigenschappen gemeen met LISP: het basiselementen is de *expressie* en een programma bestaat uit een aantal functies. Een C programma heeft altijd een functie die 'main' heet en eventueel andere functies kan aanroepen, vergelijkbaar met de 'program' declaratie in Pascal.

```
int main( void)
{
   1,2,3,4;
   "Hoedje van papier";
   return 0;
}
```

Een verschil met LISP is dat C ingebouwde datatypen onderscheidt en samengestelde datatypen kan construeren, zoals het array en de structure (in Pascal heet dat een record). In bovenstaand voorbeeld wordt de functie main() gedefinieerd die geen parameters kent (void) en een int (geheel getal) waarde retourneert, wat ze gelijk maakt aan wat in Pascal een procedure heet. De accolades zijn vergelijkbaar met 'begin' en 'end'. Iedere expressie wordt afgesloten met een puntkomma en het is toegestaan om expressies te scheiden met komma's. Het programma berekent een viertal expressies van type int, gevolgd door een string (tekst), gevolgd door het resultaat, dat je kunt opvragen met het Unix commando

```
echo $?
```

C is een *imperatieve* taal: de expressies worden sequentieel geëvalueerd (De C11 standaard ondersteunt ook *multithreaded* programmeren). De taal maakt veel gebruik van variabelen, zoals 'x' hierboven, die telkens andere waarden krijgen met de toewijzings-operator '='.

Een C programma kan worden samengesteld uit meerdere broncode bestanden en kan functies, constanten en variabelen uit andere modules gebruiken. Veel modules worden geleverd in de vorm van libraries, waarvan een aantal in de C standaarden zijn vastgelegd (zoals de afdruk functie en de arctangens in onderstaand voorbeeld).

```
#include <stdio.h>
#include <math.h>

void main( void)
{
   float        pi, (*f) (float x);

   f = atanf, pi = 4.0 * f( 1.0);
   printf( "De waarde van pi is ongeveer %f.\n", pi);
}
```

Het gebruik van de standaard libraries is niet verplicht. Binnen de kernel zijn ze niet beschikbaar omdat ze functies van het OS moeten aanroepen, zoals printf() om een string op de terminal of op papier af te drukken. Vorig voorbeeld illustreert het gebruik van de macro processor om definities van library functies, constanten en variabelen in te lezen. De preprocessor operaties worden voorafgegaan door een *spoorwegteken* alias *hekje* ('#'). De macro's zijn niet te onderscheiden van de namen van variabelen of functies.

Het voorbeeld toont hoe de variabele 'f', die een drijvende-komma parameter 'x' nodig heeft en een float waarde retourneert wordt gedeclareerd. De functie krijgt vervolgens (het adres van de) arctangens functie als waarde toegewezen en ze wordt aangeroepen om de waarde van π te berekenen en af te drukken. (In C11 kun je ook Griekse letters gebruiken) C kent geen hogere-orde functies, maar staat je toe om een functie als parameter aan een andere functie mee te geven.

Geheugenbeheer

Over geheugenbeheer onder MS-DOS zijn boeken volgeschreven. In de meeste gevallen heeft elk programma een eigen geheugenruimte, die kan worden beschouwd als een array van char variabelen, genummerd van 0 .. 2^n, met n = 16, 32 of 64. Slechts een deel van die adresruimte is daadwerkelijk voor een proces beschikbaar.

Voor de C ontwikkelaar is een klein deel van het geheugen gevuld met de *omgevingsvariabelen*, gevolgd door de binaire programmacode. Een deel is voor globale en statische variabelen en constanten. De *stack* bevat de automatische (lokale) variabelen; met behulp van het besturingssysteem kan die dynamisch groeien en krimpen; in een multithreaded applicatie heeft elke thread een eigen stack, maar deelt de globale variabelen en de heap.

De *heap* is een stuk geheugen dat dynamisch groeit onder beheer van de programmeur: de malloc() functie reserveert een aantal bytes en de free() functie retourneert ze weer. Deze functies verdelen geheugen in kleine stukjes; wanneer de arena vol is zal de C library het besturingssysteem om extra ruimte verzoeken. Lisp en Java hebben een *garbage collector* die ongebruikte objecten opruimt, waardoor die talen langzaam zijn en veel geheugen kosten.

Variabelen moeten worden gedeclareerd, waarna de compiler geheugen reserveert; constanten moeten meteen worden geïnitialiseerd, voor variabelen is dat ook toegestaan. C maakt veel gebruik van *pointers*, dat zijn variabelen die het adres van een object bevatten. Een functienaam is een soort pointer naar het adres van de machinecode.

```
{
   float        pi = 4.0 * atanf( 1.0);
   float        *pipo = &pi;
   printf( "De waarde van pi is ongeveer %f.\n", *pipo);
}
```

In het voorbeeld hierboven is 'pipo' dus een pointer naar een drijvendkomma getal, die verwijst naar 'pi'. De '&' operator betekent 'adres van' en '*' neemt de inhoud het adres waar de pointer variabele naar verwijst.

De taal C is *statically typed*, wat inhoudt dat constanten, variabelen en functie parameters van de compiler een vast absoluut of relatief adres

krijgen. De grootte van een *struct* (vergelijkbaar met het *record* type van Pascal) volgt uit optelling van de grootte van de leden plus alignment bytes, terwijl voor het *union* type (dat anders dan in Pascal geen *tag* heeft) de grootte gelijk is aan de grootte van het grootste lid.

Een recursief datatype zoals een boom of een lijst kan in potentie oneindig groot worden; daarom worden de elementen één voor één geïnitialiseerd en gebruikt men pointers om het geheel te verwerken. In onderstaand voorbeeld is een void * gebruikt voor een lijst waarvan de elementen een willekeurig type mogen hebben.

```
typedef          struct _node  /* linked list element */
{ void           *info;
  struct _node   *next;
} NODE, *NODE_P;
```

De argumenten van een C functie worden altijd *by value* doorgegeven, dat wil zeggen dat de waarde van een expressie (bij voor beeld een kopie van een variabele) wordt berekend en op de stack gezet.

Het effect van *reference parameters* wordt bereikt door een functie niet te declareren met een parameter van type int, maar int *. Als 'x' een variabele van type int is, dan wordt zo'n functie aangeroepen met &x als argument. De aangeroepen functie kan de pointer wijzigen zodat die naar een ander adres wijzigt, maar vooral ook de inhoud van die variable 'x'. Als een parameter als array is gedeclareerd, wordt een pointer naar het eerste element doorgegeven in plaats van de hele inhoud op de stack te kopiëren.

```
{
   char          hallo[] = {'W','e','l','k','o','m',
                 ' ','b','i','j',' ','d','e',' '};
   char          hcc[] = "Hobby Computer Club";
   char          b = 'a' + 1;
}
```

Strings zijn arrays van char, waarin het char type een 8-bits int is (signed of unsigned). Hierboven worden er twee gedeclareerd en geïnitialiseerd. De compiler berekent in dit geval de benodigde array grootte en sluit de tekst af met een NUL byte. Met het 'strings' commando kunt u zien welke tekstregels zich in een binair programma bevinden.

Als een array niet meteen geïnitialiseerd wordt, moet in de code het aantal elementen worden opgegeven, zodat de compiler genoeg geheugen reserveert. Als een functie een array parameter meekrijgt, dan is het niet vereist om de grootte van het array op te geven, aangezien C de indices toch niet controleert: 'a[-1]' zal door de compiler geaccepteerd worden en leidt snel tot een crashend programma.

In C worden arrays beschouwd als een alternatieve notatie voor pointers: de types 'float *' en 'float []' zijn zodoende synoniem. Het tweede element van array a kun je schrijven als a[1], a + 1, 1 + a, of 1[a]. Als je een variabele als pointer declareert, wordt er alleen ruimte gereserveerd voor een pointer, terwijl een array declaratie ruimte reserveert voor de inhoud; met malloc() reserveer je precies de benodigde hoeveelheid ruimte.

Waarom zou u C leren?

De belangrijkste reden om C te leren is dat het veel wordt gebruikt en overal op draait. Het is erg geschikt voor embedded en systeemprogrammeren. Als u C kent kost het leren van andere talen daarna minder moeite, vooral voor talen waarvan de syntax van C is afgeleid, zoals JavaScript. Voor de beginner is Pascal geschikt als eerste taal, omdat je daarmee iets minder fouten maakt en die sneller oplost, wat C een goede tweede taal maakt.

Voordelen

Er zijn veel programma's, compilers en libraries in C beschikbaar. C code is snel (bijna net zo snel als Fortran, zodat het veel voor wetenschappelijk rekenwerk gebruikt wordt) en vergt weinig geheugen (handig voor een apparaat zonder OS). Je kunt er bijna alles mee wat in assembler mogelijk is. Het heeft niet alles, maar juist de mogelijkheden die u in de praktijk meestal gebruikt. Met de vele libraries zijn de meeste functionaliteiten beschikbaar die andere talen bieden.

Nadelen

C code is beknopt en niet altijd optimaal leesbaar, maar hackers vinden dat een voordeel. Met commentaar in de code maakt u veel goed. C biedt de programmeur veel vrijheid, wat veel organisaties compenseren door bijvoorbeeld het gebruik van *goto* te verbieden. De typecontrole is sterk, maar kan door typecasting worden omzeild. De grenzen van arrays worden niet gecontroleerd, waardoor gemakkelijk *bugs* ontstaan die soms moeilijk te vinden zijn. Er zijn wel eens problemen met portabiliteit, waardoor bijv. code die voor een 16-bits machine is geschreven, niet goed werkt op 32-bits processoren.

De syntaxis is niet gemakkelijk te parsen, waardoor compilatie langer duurt.

Boeken en Compilers

C Compilers zijn beschikbaar voor de meeste 4-bits, 8-bits, 16-bits, 32-bits, 64-bits architecturen, zowel Open als Closed Source, voor embedded targets

bestaan cross-compilers. De meeste compilers ondersteunen zowel C als C++.

Om C te leren is de tweede editie van 'The C programming language' van Kernighan & Ritchie nog steeds een aanrader voor wie al enige programmeerkennis heeft. De officiële C11 standaard is te koop voor wie het exact wil weten. Voor beginners wordt “Een methode van programmeren” van Dijkstra & Feijen (ook wel: “A method of programming”) aangeraden.