ЛЕКЦИЯ 14
ПРОГРАМНИ ЧАСТИ

- Структура на програмата
- Видове програмни части
- Видове променливи
- Процедури и параметри
- Видове параметри
- Активиране на процедури
- Разпределение на паметта

СТРУКТУРА НА ПРОГРАМА

В най-простия случай една програма представлява последователност от оператори, евентуално предшествана от дефиниции на използваните елементи.

По-големите програми се разделят на части.

По-точно такова програми се сглобяват от отделно оформени програмни части и допълнително написани оператори.

Някои компилатори превеждат не цялата програма, а само отделна програмна част. Това се нарича разделна компилация.

ПРИЧИНИ ЗА ДЕЛЕНЕ

Създаването на програми, включващи отделно оформени части, се извършва с цел:

1. за да има програмата по-компактен и прегледен вид;
2. за да може при създаването то да участват повече хора и евентуално компютри;
3. за да може да се използват изработени и проверени части от други програми;
4. във връзка с ограниченията на ОП, поради които в нея могат да бъдат поместени данни и програми с определен обем.

ВИДОВЕ ЧАСТИ

ЕП предлагат 4 вида програмни части: блок, процедура (подпрограма), функция и модул.

По начина на своето определяне (писане) програмните части могат да бъдат вложени (Паскал, Алгол–60) и независими (Си, ВБ).

При втория случай някои компилатори могат да превеждат и отделна програмна част, което е известно като разделна компилация.

По начина на изпълнение програмните части биват главна (активираща) и подчинена (активирана, използвана).

ВИДОВЕ ПРОМЕНЛИВИ

Характерна особеност на програмните части е възможността във всяка от тях да бъдат определяни допълнителни променливи.

Тези собствени променливи се наричат локални променливи и са недостъпни за обхващащите и независимите по написване.

Локалните променливи се „раждат“ при започване на изпълнението на дадена част и „умират“ в края на нейното изпълнение.

За обменяне на информация с другите части се използват нелокални променливи, които биват глобални, общи и параметри.

ОБЛАСТ НА ДЕЙСТВИЕ

Тъй като всяка част има право на собствени имена възниква възможност за използване на еднакви имена.

В този случай трябва да се въведе правило за определяне кое име какво означава.

Това правило е известно като „видимост“ или „област на действие“ на имената.

При вложено описание на частите, имената от обхващащия блок могат да се използват и във вътрешния блок (глобални), освен когато те са дефинирани повторно (локални).

ГЛОБАЛНИ ИМЕНА

При езиците с независимо описание на програмни части е възможно променлива от една част, да бъде използвана в друга част.

Така при разделна компилация няма да бъде известен типа на променливите, дефинирани в друга програмна част.

За разрешаване на проблема в такива езици освен дефиниция на променлива (мястото, където за нея ще бъде отделена ОП) се използва и декларация, посочваща типа на глобалните за частта променливи.

ОБЩИ ПРОМЕНЛИВИ

Вместо глобални променливи, които имат еднакви имена във всички програмни части, някои езици (Фортран) предлагат използване на общи променливи.

Те имат различни имена във всяка част, но се разполагат на едно и също място в ОП.

Общите променливи изискват познания за представянето на величините в ОП и дават възможност за някои „трикове“, поради което са премахнати в съвременните ЕП.

Последна редакция 16 Септември 2004 г.

ПРИМЕРИ

Паскал: По маниер на писане програмните части са вложени една в друга. Всяка част има право да дефинира свои собствени (локални) елементи. Използването на недефинирано име означава (глобален) елемент от обхващащата програмна част.

Си: Освен правилата на Паскал с `external` може да се декларират глобални имена.

ВБ: Клаузите Public/Private определят дадено име като глобално (използваемо вън от)/ локално (неизвестно вън от).

СЕЦИАЛЕН СЛУЧАЙ

Правилото, че локалните променливи се „раждат“ при стартиране на програмна част и „умират“ при нейното завършване има и свои отрицателни черти.

Затова повечето ЕПВР дават възможност част от променливите, наречени статични, да запазват своите текущи стойности между две изпълнения на съответната част.

Такива променливи обикновено се „раждат“ при стартиране на програмата и „умират“, когато нейното изпълнение завърши.

ПРИМЕР: СТАТИЧНИ

Паскал: няма такива променливи.

Си: Променливите, дефинирани чрез `static` <тип> <име> са статични.

ВБ: Променливите, дефинирани като `static` <име> `[As` <тип>`]` са статични;

В процедура (функция), чието име се предшества от `static`, всички дефинирани променливи са статични.

БЛОКОВЕ

Като програмна част блокът е най-елементарната програмна част.

Не всички ЕПРВ предоставят блокове.

Блоковете винаги са вложени по своя маниер на написване.

Единствената разлика между съставен оператор и блок е в правото при писане на блок да може да бъдат дефинирани нови (локални за блока) променливи.

ПРОЦЕДУРИ

Процедурата е програмна част от по-високо равнище спрямо блока.

Тя се дефинира на едно място в програмата, но може да се използва във всяка друга точка от написаната програма.

Ползата от процедурите проличава най-ясно, когато те могат да се параметризират.

Функциите са особен род процедури.

Всеки ЕПВР предоставя наготово определен брой „стандартни“ процедури и функции.

ФОРМАЛНИ ПАРАМЕТРИ

За да параметризираме текста на една процедура в него трябва да отбележим участъците, подлежащи на промяна.

Отчитайки, че тези участъци най-често ще представят данни, става ясно че те ще бъдат представяни от параметризиращи величини и те най-естествено ще са променливи.

Този род променливи, необходими за написване на една процедура, се наричат формални параметри на процедурата.

За всеки от тях трябва да бъде посочен тип.

ФАКТИЧЕСКИ ПАРАМЕТИРИ

При всяко използване на една процедура формалните параметри се заменят с действителните величини, с които се провеждат параметризираните изчисления.

Тези действителни величини се наричат фактически параметри на процедурата.

За използване на процедура в ЕПВР трябва да предоставя оператор за използване на процедура и ред за съответствие между фактическите и формалните параметри.

ВИДОВЕ ПАРАМЕТРИ ❶

В зависимост от посоката на предаване на информация параметрите се делят на входни (начални данни за процедурата) и изходни (резултат от нейната работа).

Функциите са особен род процедури с допълнителен изходен параметър, свързан с името им и наричан тяхна стойност.

Тази разлика рефлектира в начина на използване: при процедурите имаме специален оператор, докато функциите се използват при писането на изрази.

ВИДОВЕ ПАРАМЕТРИ ❷

Параметрите на процедурите се различават **по механизма за съпоставяне** на фактическите и формалните параметри.

По този показател параметрите биват:

1. **стойности** (**Паскал**, **Си**, **ВБ**): в процедурата се получава **стойност без сведения за произход**;
2. **променливи** (**Паскал**, **ВБ**): в процедурата е **известен произходът** (име на променлива);
3. **процедури** (функции) (**Паскал**): процедурата получава **име на друга процедура** (функция);
4. **стойност-резултат** (**Фортран**);
5. **име** (**Алгол-60**): текстова замяна.

ДЕФИНИЦИЯ НА ПРОЦЕДУРА

За да се дефинира една процедура трябва:

1. да се посочи нейното **име**;
2. да се посочат **броят**, **имената**, **типът** и **видът** на всички **формални параметри**;
3. **при функция** да се посочи и **типът на резултата**;
4. да се напише **кодът на процедурата**.

За да се използва коректно една процедура е достатъчно да бъдат известни ❶, ❷ и ❸, които формират и **декларацията на** една **процедура** (функция) при разделно писане.

ПРИМЕР: ПАСКАЛ ❶

Процедури (функции) **се дефинират** в специален раздел **в началото на** всяка програмна **част и** могат да **се използват** само **в нея и в нейните вложени** части, стига в тях да няма нова дефиниция със същото име.

Дефиницията има вида:

`PROCEDURE` <име> [`(`<списък формални парам.>`)`]`;` [<дефиниции>] <оператор>`;`

`FUNCTION` <име> [`(`<списък форм. пар.>`)`] `:` <тип> `;` [<дефиниции>] <оператор>`;`

ПРИМЕР: ПАСКАЛ ❷

Формалните параметри в списъка **са** разделени с точка и запетая (`;`) **описания**

[<вид>] <име> `:` <тип>

Когато липсва <вид> се приема, че е описан формален **параметър-стойност**.

В останалите случай за <вид> се посочва: `VAR` за параметрите-**променливи**, `PROCEDURE` за параметрите-**процедури** и `FUNCTION` за параметрите **функции**.

Използваният <тип> трябва да е **дефиниран** предварително **в раздела за типове**.

ПРИМЕР: ПАСКАЛ ❸

В програмния код на **функция** трябва да има поне един **оператор**, който **да присвои стойност на името** на функцията (резултат).

Процедурите и функциите се **използват в** програмната **част**, в който са дефинирани, като **след тяхното име** в кръгли скоби се запише **списък на фактическите параметри**.

Фактическите параметри трябва **по брой и тип да съответстват на формалните** като съпоставянето е **по позицията на писане**.

ПРИМЕР: СИ ❶

Програмата е съвкупност от функции, **една** от които носи **името** `main`.

Не се разрешава определяне на **процедури**, но **типът на функция** може да бъде `void`, което **означава липса на резултат**.

Дефиницията на функция има **вида**

<тип> <име> `(`[<списък от формални параметри>]`)` <блок>

Стойността на дадена функция **се определя** от израза **в оператор** `return` <израз>.

ПРИМЕР: СИ ❷

Формалните параметри в списъка **са** разделени със запетая (`,`) **описания**

<тип> <име> [`()`]

Всички параметри са стойности, защото езикът поддържа **явен тип указател** към, който може да се използва **при параметри, чиито произход е от значение**.

Вместо по стойност **масивите се предават чрез указател** към своя първи елемент.

Проверка за съответствие по брой и тип се извършва **само при наличие на прототип**.

ПРИМЕР: ВБ

Дефиницията на процедура има вида

```
Sub <име> (<списък формални парам.>)
 <оператори> ' вкл. Exit Sub
End Sub
Function <име> (<форм. пар.>) As <тип>
 <оператори> ' вкл. Exit Function
 [Let] <име> = <израз> ' определя резултата
End Function
```

Параметрите се разделят **със запетаи** (`,`):

[`ByVal` | `ByRef`] <име>[`()`] [`As` <тип>]

`ByVal` – по стойност, `ByRef` – по позоваване.

ПРОИЗВОЛЕН БРОЙ ПАРАМЕТРИ

Обикновено ЕПВР изискват **броят на формалните** параметри да бъде **фиксиран** и всяко **използване** да бъде **винаги** със същия **този брой фактически** параметри.

Малко езици дават възможност за писане на процедури с **произволен брой параметри**.

Дефиницията на подобна процедура **съдържа имената на задължителните параметри**.

Следващите (произволен брой) параметри **са достъпни** в кода **по специален начин**.

ПРИМЕРИ

Си: <тип> <име> `(`<списък>`, ...)` {`;`|<блок>}

При дефиницията в <блок> **за достъп до** безименните **параметри се използват**:

`va_list` **специален тип** за достъп до параметри;
`va_start` **начало на достъпа** до параметрите;
`va_arg` за **извличане на** поредния **параметър**;
`va_end` **край на достъпа** до параметрите.

ВБ: **последният формален параметър** може да бъде обявен като `ParamArray`. Това означава **тип** `Variant` **със стойност масив**, съдържащ допълнителните **параметри**.

ПАРАМЕТРИ ПО ИЗБОР

Само ВБ предоставя такива параметри.

Декларират се в списъка с параметри **чрез**

`Optional` [`ByVal` | `ByRef`] <име>[`()`]
[`As` <тип>] [`=` <начална стойност>]

Началната стойност може да се посочи само **чрез константен израз** и **се използва при отсъствие** на фактически параметър.

Липсата на съответстващия **фактически параметър** може да **се констатира** и **чрез** стандартна **функция** `IsMissing`.

ИЗПОЛЗВАНЕ

Дефинираните в една програма **процедури** могат да **се използват чрез** прост **оператор за активиране на произволно място**, където е „видимо“ (т. е. известно) тяхното име.

Използването на процедура става като след нейното **име се посочи списък, съдържащ** съответстващите **фактически параметри**.

Обикновено списъкът се поставя **в скоби**, а **параметрите** се отделят **със запетая** (,).

Функциите се използват само **в изрази** и то чрез скоби. **Заместват се със своя резултат**.

СЪОТВЕТСТВИЕ

Фактическите параметри **се свързват със** съответния **формален** параметър **по реда на** своето **писане** (**по позицията** си) и трябва да отговарят на следните **правила**:

ФОРМАЛЕН параметър	**ФАКТИЧЕСКИ** параметър
стойност	**израз** от същия тип
променлива (**VAR**) / позоваване (**ByRef**)	проста **променлива** или **елемент** на структура
процедура (функция) в **ПАСКАЛ**	**име** на известна процедура (функция)

КЛЮЧОВИ ПАРАМЕТРИ

Освен съответствие по позиция **ВБ** дава възможност при използване на процедура (функция) всеки **фактически параметър явно** да **посочва името на** своя **съответен формален** параметър.

Такива параметри се наричат **ключови**.

Ползата от такива параметри **се вижда най-добре**, когато **фактическият параметър** на една процедура **е еднакъв при голяма част** от случаите **на** нейното **използване**.

МОДУЛИ

Някои ЕПВР (**Си**, **ВБ**, **Модула**) предлагат **най-високото равнище** на програмни части, наречено **модул**.

Всеки модул е отделен файл и **съдържа** дефиниции на определен брой **процедури и данни**, необходими за тяхното използване.

Компилаторите, осигуряващи превод на отделна програмна част (**Си**), **работят** само **на равнище модул** (**разделна компилация**).

Но деленето на програмата **на модули не е свързано с разделната компилация** (**ВБ**).

ВИДОВЕ ПАМЕТ

Основната идея на ЕПВР **е** при писане на програми **да забравим за** архитектурата, присъща на **даден компютър**.

Тази идея е **прекрасна**, **но** уви **не е реална**.

Поради това **някои ЕПВР** си **позволяват** да предоставят **конструкции**, чрез които **да подскажем** на превеждащата програма **някои наши желания**.

Основният елемент, подлежащ на конкретизиране, **е нееднородната ОП**.

ПРИМЕР: ПАСКАЛ

Паскал предоставя **два вида** променливи: **обикновени и динамични**.

Памет за обикновените променливи **се отделя при започване** на изпълнението на съответната програмна част (**началната** им **стойност** е **произволна**), **и се освобождава при завършване** на изпълнението.

Памет за динамични променливи **се заделя и освобождава по заявка** на програмата.

Динамичните променливи на **Паскал се представят чрез указатели**.

ПРИМЕР: ПАСКАЛ (прод.)

Динамична променлива се **дефинира** чрез <име> `:` `^`<тип>`;` , при което **вместо** нова **променлива се създава указател** към <тип> именован с <име>.

Памет за променлива, чиито адрес става стойност на указателя, **се заделя** чрез `NEW(`<име>`);` **и освобождава** ръчно с `DISPOSE(`<име>`);`(„раждане“ и „умиране“).

Стойността на съществуваща динамична променлива **се цитира чрез** <име>`^`, а **единствената константа** за указател **е** `NIL`.

ПРИМЕР: СИ

Си поддържа **явно** типове **указател към**.

За динамично **заделяне и освобождаване** на памет **се използват** стандартните функции:

`void * malloc(`size_t <размер>`)`,

`void * calloc(`size_t <бр.>`,` size_t <рзм.>`)`

и `void free(void *` <указател>`)` .

Прототипите на тези функции, заедно с определение на **празния указател** `NULL`, са записани **в стандартните библиотечни файлове** `stdlib.h` и `malloc.h`.

ПРИМЕР: СИ (прод. 1)

Променливите, дефинирани **във функция**, са **локални** за нея, а тези **вън от функция** – **глобални** за цялата програма.

Глобалните променливи **са достъпни** пряко **в модула, в който са дефинирани**.

Глобалните променливи **се раждат с** начална **стойност 0 при стартиране** на програмата и **умират при** нейното **завършване**.

За използване на **глобална променлива от друг модул** тя трябва да бъде **декларирана чрез** `extern` <тип> <име>.

ПРИМЕР: СИ (прод. 2)

Всеки **ЦП** притежава **регистри**, които функционират като **свръх бърза памет**.

Поместването на стойностите **на често използваните променливи в регистри** на ЦП **ускорява** изпълнението на програмата.

Си решава нещата като **дава възможност пред** дефиницията на **локална** променлива да бъде записано `register`.

Компилаторът е длъжен да **моделира такива променливи в регистри** на ЦП (**стига да има** свободни такива) вместо в ОП.

ПРИМЕР: ВБ

Променливите, дефинирани **в текста на** една **процедура**, са **локални** за нея, **а вън от** този **текст – глобални на модулно равнище**.

Глобалните стават общо достъпни когато се дефинират с `PUBLIC` вместо с `DIM`.

Процедурите и функциите са глобални за цялата програма, **освен когато** пред името им **е записано** `PRIVATE`.

Динамично създаване на нови **екземпляри**, достъпни чрез обектовите променливи на ВБ (указателите), **става** чрез `NEW` <клас>.

ПРИМЕР: ВБ (прод.)

Локалните променливи на процедура **се раждат с фиксирана стойност** (`0`, `0.0`, `""`, `Empty`, `Nothing`) **при** нейното **започване** и **умират заедно** със създадените от нея **динамични екземпляри при завършване** на тази процедура (функция).

Глобалните променливи на модулно и програмно ниво **се раждат при стартиране на програмата** и **умират в нейния край**.

Стойността на глобалните също е фиксирана.

СТАТИЧНИ ПРОМЕНЛИВИ

Правилото за „прераждане“ на локалните променливи на процедурите **решава редица проблеми, но поражда нови**.

Често е **полезно стойността на** една локална **променлива** да бъде **запазена от края на** едно **изпълнение на процедура към началото на следващото** изпълнение.

Използването на **глобална променлива** за подобна цел **не е добро решение**.

Някои езици предлагат **по-добро решение**: използване на **статични променливи**.

ПРИМЕРИ

Паскал: няма статични променливи.

Си: пред дефиницията на статична локална променлива се записва `static`.

Записването на `static` пред името на функция прави тази функция локална.

ВБ: статични променливи се дефинират само в процедури, като вместо `Dim` се използва `Static`.

Записването на `Static` пред името на процедура означава, че всички нейни локални променливи са статични.

СТРАНИЧНИ ЕФЕКТИ

Променянето на стойността на глобална променлива **и на параметър** при изпълнението на дадена процедура (функция) се нарича неин **страничен ефект**.

Активирането на процедура е единичен **оператор**, а **и някои** от тях **се използват само заради** своите **странични ефекти**.

Функциите се използват при изчисляване на **изрази**, поради което **страничните ефекти** на една функция могат да доведат до **проблеми**, **свързани с реда** на изчисляване.

РЕКУРСИВНИ ПРОЦЕДУРИ

Процедура (функция), **която** при своето изпълнение **използва** самата **себе си** се нарича **рекурсивна**.

Рекурсията бива **два вида**:

❶ **Пряка** (явна): **А** използва **А** (т. е. себе си);

❷ **Косвена** (неявна): **А** използва **В**, а **В** – **А**.

Рекурсията позволява елегантно описание на редица алгоритми.

Рекурсивните процедури (почти) **не** бива да **използват статични** локални **променливи**.

Рекурсията е еквивалентна на итерацията.

РЕКУРСИЯ (продължение)

В някои ЕПВР (**Фортран**) **рекурсията е забранена**, поради **трудната реализация**.

Всички процедури и функции в **Паскал**, **Си** и **ВБ** допускат рекурсивно използване.

При **Паскал** и **ВБ** всяко използване на **името на функция в дясната част** на оператор за присвояване води до **ново** (**рекурсивно**) **активиране** и трябва да се придружава от списък с фактически параметри.

В лявата част на същия оператор **името** на функцията **определя** нейния **резултат**.

ПРОБЛЕМИ НА ПАСКАЛ

Писането на **косвено рекурсивни** процедури **при Паскал** поражда **конфликт** с неговите правила: **всеки** елемент **да се дефинира преди** да бъде **използван**.

За **разрешаване на конфликта** е предвидена служебната дума `FORWARD`, която **замества дефиницията** (текста) на първата процедура (функция), защото **заглавието** (**с параметрите**) **е достатъчно за** нейното **използване**.

Формалните параметри не се записват повторно при следващата дефиниция на кода.

БЛАГОДАРЯ ВИ ЗА ВНИМАНИЕТО!

БЪДЕТЕ С МЕН И В СЛЕДВАЩАТА ЛЕКЦИЯ, КОЯТО ЩЕ НИ ОТВЕДЕ В НЕВЕРОЯТНИЯ СВЯТ НА МАКРОАПАРАТА