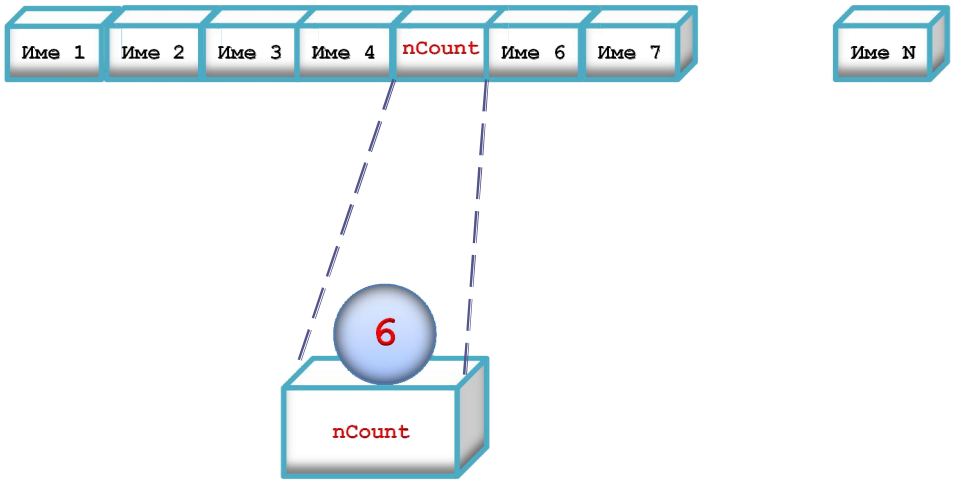
Типове данни и операции с тяхна

При изпълнението на програма се извършват определени действия над данните, дефинирани в програмата.Тези данни могат да бъдат постоянни (константи) или изменящи се (променливи).

Тези данни най-често бива числова и текстова. За съхраняването и в паметта на компютъра е необходимо различно по обем място. Размерът на това място се определя от типът на данните. Нека да си представим, че паметта на компютъра е мрежа от празни кутийки, които чакат да бъдат напълнени с информация. След като дадена кутийка бъде напълнена, информацията, която тя съдържа може да бъде използвана многократно, също така може да бъде променяна или просто изтрита.



Фиг. 1. Абстрактно представяне на данните в паметта на ПК

Това е и главната причина променливите да бъдат наричани по този начин, те могат да съдържат информация, която се съхранява само и единствено по време на изпълнението на програма, след това цялата информация бива директно изтривана автоматично. Друго нещо, което е много важно да се запомни, е че променливите могат да съдържат различни типове информация, което определя и типът на данните в тях.

Езиците от високо ниво позволяват структуриране на данните по тип.

Това структуриране определя размера на паметта, отделена за всеки елемент от данни, и позволява де се генерира най-ефективния код за изпълнение на операции C++ този елемент от данни. Всяка променлива в C, преди да се използва в програмата, трябва да се дефинира (опише). Дефинирането на променливите има следния синтаксис:

```
<ТипНаПроменлива> <ИмеНаПроменлива>;
```

За **ТипНаПроменлива** се използва ключова дума (думи) за един от допустимите в езика типове от данни или име на тип дефиниран от потребителя. За **ИмеНаПроменлива** се използва идентификатор

отговарящ на определени изисквания.

На фиг. 2 са представена една примерна класификация на данните.

Фиг. 2. Видове данните

- **Скаларни** са типовете данни, които се състоят от една компонента (число, знак и др.).
- **Съставни** типове са онези типове данни, компонентите на които са редици от елементи.
- **Типът** указател дава средства за динамично разпределение на паметта.

Типовете данни представляват множества от стойности, които имат еднакви характеристики. Това са:

- Име – например **float**;
- Размер (колко памет заемат) – например 4 байта;
- Стойност по подразбиране например 0.0.

Базовите типове данни в C/C++ се разделят на следните видове:

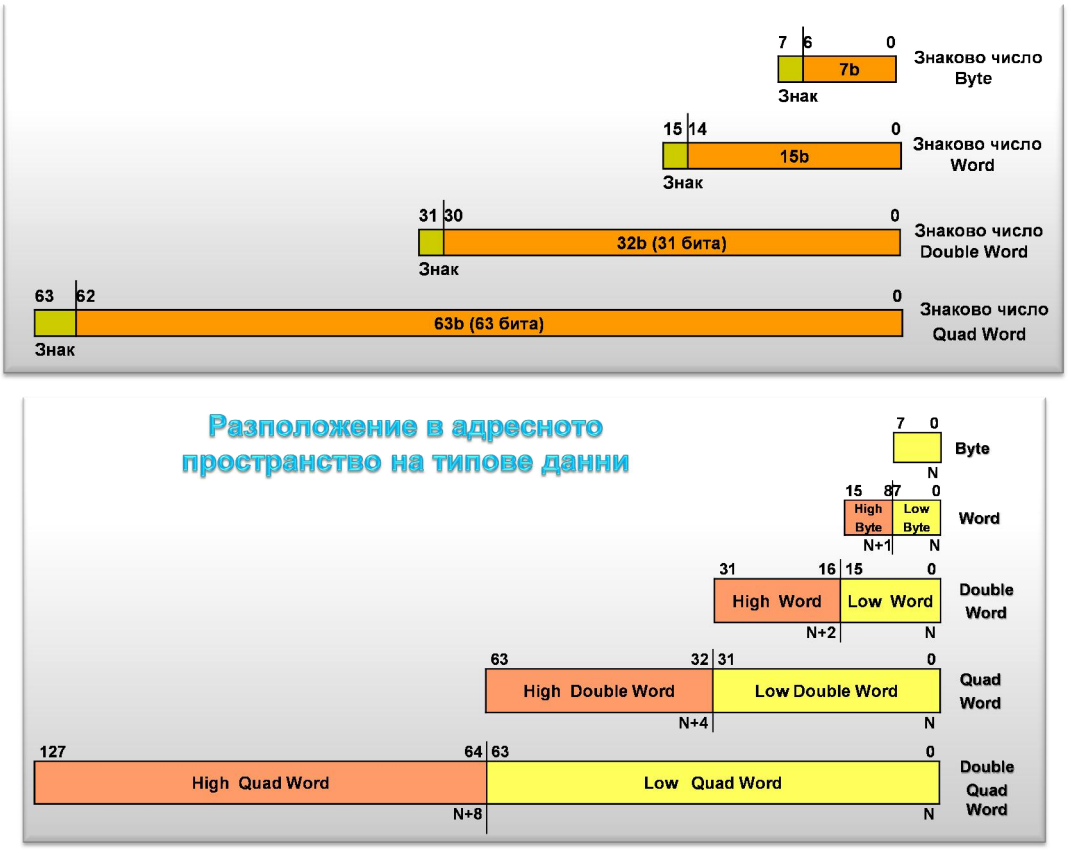
- Целочислени типове – **byte, int;**
- Реални типове с плаваща запетая – **float, double;**
- Булев тип – **bool;**
- Символен тип – **char;**
- Структури – **struct**
- Обединения – **union**
- Изброяем тип – **enum**
- Логически тип – **bool**

Аритметичните типове **char**, **int**, **float**, **double**,**byte**, **bool** типът **void** и функционалния тип са основни типове. Те са предварително дефинирани в езика и се поддържат от неговото ядро. Чрез аритметичните типове **int**, **char**, **float** и **double** се представят числените данни (цели и дробни). Тези типове могат да се използват

в комбинация с някой от модификаторите signed, unsigned, short и long, които доопределят някои техни аспекти.

Цели числа

Повечето числа, които са записани като десетични целочислени числа, се съхраняват в паметта на компютъра като двоични числа със или без знак. В зависимост от големината на числото за представянето му са необходими различен брой цифри (битове). В изчислителните системи за представяне на числата обикновено се отделя фиксиран брой битове (байтове). Това означава, че в областта, определена за представяне на дадено число може да се изобрази стойност в ограничен диапазон.



Фиг.3 Стандартни типове цели числа без знак

Максималната стойност (число), която може да се представи в област с ***n*** бита се определя от броя на различните комбинации от '0' и '1', които могат да се съставят с ***n*** двоични цифри. Стойността на най-старшият бит (този който се намира най-отляво) определя знака на числото. Стойност 0 указва положително число а 1 отрицателно. Диапазона на числата е $\pm 2^{n-1}$ В програмният език C/C++ целите числа се съхраняват в променливи от тип **char**, **int** и техните модификации: **byte**, **short int**, **long int**, **unsigned int**, **unsigned short int**, **unsigned long int** и **long long**.

Типът **char** е 8-битов. Той може да бъде използван както за съхраняване на текстова информация (ASCII символи - основно предназначение), така и за съхраняване на целочислена данни със знак. Това означава, че броят на възможните му стойности е 2^8,

т.е. 256 възможни стойности общо, като те могат да бъдат само както положителни, така и отрицателни. Минималната стойност, която може да се съхранява е -128 (-2^7), а максималната 127 (2^7-1). Стойността по подразбиране е числото 0.

- Типът **byte** е 8-битов беззнаков (**unsigned**) целочислен тип. Той също има 256 различни целочислени стойности (2^8), но те могат да бъдат само неотрицателни. Този тип е въведен в C++ в ANSI C не е дефиниран и се образува декларацията **unsigned char**. Стойността по подразбиране на типа **byte** е числото 0. Минималната му стойност е 0, а максималната 255 (2^8-1).
- Целочисленият тип **short int** е 16-битов тип със знак. Минималната стойност, която може да заема е -32768 (-2^{15}), а максималната -32767 ($2^{15}-1$). Стойността по подразбиране е числото 0.
- Типът **unsigned short int** е 16-битов беззнаков тип. В програмирането е познат като тип WORD. Минималната стойност, която може да заема, е 0, а максималната 65535 ($2^{16}-1$). Стойността по подразбиране е числото 0.
- Типът **int** е най-често използваният тип в програмирането. Обикновено програмистите използват **int**, когато работят с цели числа, защото този тип е естествен за 32-битовите микропроцесори и е достатъчно "голям" за повечето изчисления, които се извършват в ежедневието.
  Той е 32 - битов число със знак. и приема стойности в интервала от -2147483648 (-2^{31}) до 2147483647 ($2^{31}-1$).
- Типът **unsigned int** е 32-битов беззнаков тип. Стойността по подразбиране е числото 0 и приема стойност в интервала от 0 до 4294967295 ($2^{32}-1$). Типът **unsigned int** е еквиваленте на типа **DWORD**.
- Типът **long** е 32 битов тип и съвпада по параметри със типът int в 32-ботовите компилатори. Използва се за съвместимост с по-старите версии на компилаторите.
- Типът **long long** е 64-битов знаков тип със стойност по подразбиране 0L. Минималната стойност, която може да заема типът **long long** е -9 223 372 036 854 775 808 (-2^{63}), а максималната му стойност е 9 223 372 036 854 775 807 ($2^{63}-1$).
- Най-големият целочислен тип е типът **unsigned long**. Той е 64-битов беззнаков тип с минималната стойност 0 и максималната 18 446 744 073 709 551 615 ($2^{64}-1$).

Реални числа

В компютърните системи реалните числа се представят по два начина – с фиксирана точка и с плаваща точка.

Реалните типове в C++ представляват реалните числа, които познаваме от математиката. Те се представят чрез плаваща запетая (floating-point) и биват **float** и **double**.

При запис на числата с **фиксирана точка** в двоичен формат се предполага точно определено място на позиционната точка в полето за запис на числата. Както бе показано по-горе целите числа със знак могат лесно да бъдат представени посредством двоично число,

но остава нуждата от представяне на реалните числа с десетична запетая. За представянето на едно реално число с двоичен код, то се разделя на две групи, съответно степен на числото 10 и коефициент на умножение. Това представяне е известно като представяне с плаваща запетая или научен формат на числата [8].

Например числото 3.14 се представя като: 314 x 10^{-2}. Следователно за представянето на едно реално число са необходими две целочислени числа със знак. В една потребителска програма реалното число 31.4 се записва или като 31,4 или като 3.14e+001, където e+001 е 10^{+1}. В паметта реалните числа се записват във формата показан на фиг. 4. Представянето на числата е според зависимостта: $-1^{s} M\ 2^{E}$ където:

s е с размер от един бит и задава знака на числото;

M – мантиса на числото ;

E – експонента задаваща степента на 2.

Фиг. 4. Представяне на реално число в паметта на ПК

(`s` -знак на числото, `exp` - експонента, `frac`- мантиса)

Типът данни `float` се използва за представяне на реални числа с единична точност. Той заема в паметта място от 32 бита е с размер като за мантисата се отделят 23 бита а за експонентата 8. Разглежданият тип има точност до 7 десетични знака (останалите се губят). Например числото `0.123456789` ако бъде записано в типа `float` ще бъде закръглено до `0.123457`. Диапазонът на стойностите, които могат да бъдат записани в типа `float` (със закръгляне до точност 7 значещи десетични цифри) е от `3.4E-38` до `3.4E+38`.

Вторият реален тип с плаваща запетая в езика C++ е типът `double`. Той се нарича още реално число с двойна точност (double precision real number) и представлява 64-битов тип със стойност по подразбиране 0.0d или 0.0D и точност от 15 до 16 десетични цифри след дробната запетая.

Числата с двойна точност заемат 64 бита в паметта, от които 11 за експонента и 52 за мантиса и приема стойности в диапазона от -1.79769e+308 до 1.79769e+308.

Най-голямото дробно число в C `long double` е с размер от 80 бита и се представя с 15 бита за експонента, и 63 бита за мантиса.

Булев тип

Булевият тип се декларира с ключовата дума **`bool`**. Той има две стойности, които може да приема – **`true`** и **`false`**. Стойността по подразбиране е **`false`**. Използва се най-често за съхраняване на резултата от изчисляването на логически изрази.

Спецификатори на тип и модификатори

С поддържа следните модификатори на тип:

signed - цяло със знак (по подразбиране)

unsigned - цяло без знак

short - къс формата на символи от тип **int**

long - дълъг формат

Тези модификатори се комбинират с основните типове и променят размера им. Например тип **int** може да се модифицира по следният начин:

```
signed int i;     // цяло със знак
signed i;         // цяло със знак
int i;            // цяло със знак
unsigned  int i; // цяло без знак
unsigned  i;      // цяло без знак
short int i;      // цяло със знак къс формат
short i;          // цяло със знак къс формат
long int i;       // цяло със знак дълъг формат
long i;           // цяло със знак дълъг формат
```

Трябва да се отбележи, че типът може да се изпусне само когато той е **int**. За останалите типове е задължително той да съществува.

Пример:

```
long float  k;
long double d;
```

В C++ важи зависимостта на размера и допустимия обхват на различните типове данни от конкретния хардуер и компилатор. Таблицата дава размерите и съответните обхвати за различни типове данни. При някой компилатори определени типове се приемат но не се изменя обхвата им.

Таблица 1: Размери и обхвати на типовете данни в C

Тип	Размер (битове)	Обхват
bool	8	true,false
unsigned char	8	0 ÷ 255
char	8	-128 ÷ 127
enum	16	0 ÷ 32767
unsigned short	16	0 ÷ 65535
short int	16	-32768 ÷ 32767
unsigned int	16 или 32	0 ÷ 65535 0 ÷ 4294967295
int	16 32	-32768 ÷ 32767 -2147483648 ÷ 2147483647
unsigned long	32	0 ÷ 4294967295
long int	32	-2147483648 ÷ 2147483647
float	32	3.4E-38 ÷ 3.4E+38
double	64	1.7E-308 ÷ 1.7E+308
long double	80	1.7E E-308 ÷ 1.7E+308
long long	64	-9 223 372 036 854 775 808 до +9 223 372 036 854 775 807
wchar_t	16	0 ÷ 65535
pointer	32 или 64	Размера се определя от вида на компилатора

Забележка: Размера на типовете int, long double е различен при различните компилатори. За да можете да определите

Пример:

```
#include <iostream>  // Входно/изходни функции
#include <iomanip>   // Входно/изходни манипулатори
#include <math.h>    // математически функции
#include <windows.h> // Системни функции
using namespace std;

int main(int argc, char* argv[]){
 cout << "   data type          " << "byte"  <<  "      "
      << "    max value  "        << endl;
 cout << "bool               =  "<< sizeof(bool) << "         "          << fixed
      << setprecision(2)  << (pow(2,sizeof(bool) * 8.0) - 1)          << endl;
 cout << "char               =  " << sizeof(char)  << "         "        << fixed
      << setprecision(2)   << (pow(2,sizeof(char) * 8.0) - 1)          << endl;
 cout << "short int          =  " << sizeof(short int) << "         "    << fixed
      << setprecision(2)   << (pow(2,sizeof(short int) * 8.0 - 1) - 1)  << endl;
 cout << "unsigned short int =  "<< sizeof(unsigned short)<< "         " <<fixed
      << setprecision(2)   << (pow(2,sizeof(unsigned short) * 8.0) - 1) << endl;
 cout << "int                =  " << sizeof(int)        << "         "   << fixed
      << setprecision(2)   << (pow(2,sizeof(int) * 8.0 - 1) - 1)        << endl;
 cout << "unsigned int       =  " << sizeof(unsigned int)  << "         "<< fixed
      << setprecision(2)   << (pow(2,sizeof(unsigned int) * 8.0) - 1)   << endl;
 cout << "long int           =  " << sizeof(long int)  << "         "    << fixed
      << setprecision(2)   << (pow(2,sizeof(long int) * 8.0 - 1) - 1)   << endl;
 cout << "long long          =  " << sizeof(long long)  << "         "   << fixed
      << setprecision(2)   << (pow(2,sizeof(long long) * 8.0 - 1) - 1)  << endl;
 cout << "unsigned long int  =  " << sizeof(unsigned long) << "         "<< fixed
      << setprecision(2)   << (pow(2,sizeof(unsigned long) * 8.0) - 1)  << endl;
 cout << "float              =  " << sizeof(float) << "         "        << fixed
      << setprecision(2)   << (pow(2,sizeof(float) * 8.0 - 1) - 1)      << endl;
 cout << "double             =  " << sizeof(double)  << "         "      << fixed
      << setprecision(2)     << (pow(2,sizeof(double) * 8.0 - 1) - 1)   << endl;
 cout << "long double        =  " << sizeof(long double)  << "         "  << fixed
      << setprecision(2)   << (pow(2,sizeof(long double) * 8.0 - 1) - 1) << endl;
 system("pause");
 return 0;
}
```

Дадената програма позволява да се видят размерите на типовете данни поддържани от компилаторите. Не е необходимо да се мъчите да разберете програмата защото в нея има оператори които все още непознавате. За повърхностно запознаване ще направим малък разбор на програмата. Оператора `sizeof()` връща байтовете заемани от подаденият и параметър в паметта. Функцията `pow(x,y)` връща стойността на `x` повдигната на степен `y`, (`xy`) и е декларирана в заглавния файл `<math.h>`. Манипулаторите **`fixed`** и `setprecision()` са достъпни от заглавния файл `<iomanip>`. Първият манипулятор – `fixed`, задава фиксиран формат за извеждане на числата във входно/изходния поток. Манипулятора `setprecision(n)` извежда `n` знака след дробната запетая. Максималната стойност на нитовете данни се изчисляват по формулата:

- За типове данни със знак

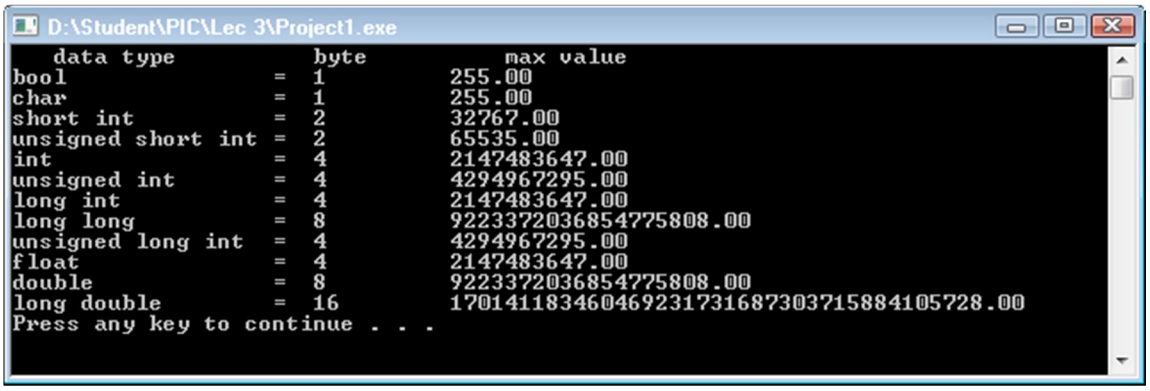
$$\mathbf{MinValType} = -2^{(b*8-1)} \qquad \mathbf{MaxValType} = 2^{(b*8-1)} - 1$$

където b е броят байтове заемани от типа в паметта. Стойността на b се умножава по 2 за да се получи размера в битове. От тази стойност се изважда 1 за да се отчете стойността на знаковия бит. Получената стойност е степента на 2 задаваща най-голямото абсолютно число, което може да бъде записано с даденият тип данни.

- За типове данни без знак

$$\mathbf{MinValType} = 0 \qquad \mathbf{MaxValType} = 2^{(b*8)} - 1$$

Резултатът от изпълнението на програмата е показан на фигурата по-долу



Модификатор signed

Освен трите модификатора дефинирани в ANSI C - **long**, **short** и **unsigned**, C/C++ поддържа модификаторите **signed**, **const** и **volatilue** (всичките включени и в стандарта ANSI). Модификаторът **signed** явно обявява, че стойност се съхранява като стойност със знак. Например когато при компилация се избере опция за работа с тип **char** без знак (**unsigned char**), то ако в програмата е необходимо за някоя променлива да се работи със знак, модификаторът дава възможност тя да се обяви като **signed char**. Модификатор **signed**, използван без тип се приема за **signed int**.

В ANSI C няма данни от булев тип (**bool**). Изрази, които работят с булеви стойности, интерпретират стойността нула като "неистина" (**false**), а останалите като "истина" (**true**). Освен изброените типове данни, C/C++ поддържа още и тип **enum**, но за разлика от Pascal, това са просто предварително присвоени цели константи и те са напълно съвместими с всички останали съставни типове. При компилаторите на C++, е въведен булев тип **bool**, който е еквивалентен на **boolean** в Pascal .

Тип void

Според дефинициите на K&R, всяка функция трябва да връща стойност. Ако не е дефиниран тип за функцията, тя се приема от тип **int**. C++ поддържа дефинирания в стандарта ANSI тип **void**. Той се

използва за обявяване на функция, която не връща стойност. Аналогично, с помощта на **void** може да бъде дефиниран празен списък от параметри на функция, която не използва параметри.

Например:

```
#include <stdio.h>

void PutMsg(void)   {
   printf("Привет!\n");
}

int main(void ) {
    PutMsg();
    return 0;
}
```

Като специална конструкция може да се използва операторът (тип) с тип **void**, за да се посочи явно, че се пренебрегва резултатът от работата на функция. Например за да се реализира пауза до натискане на произволен клавиш, може да се използва конструкцията - **(void) getch();** Може да се декларира и указател към тип void. Създаденият указател е указател към произволен тип (така не е нужно да се знае типът). На указател тип void може да се присвои стойността на всеки друг указател и обратно. Не е позволено използването на оператора * (съдържанието на адрес...), тъй като при void не е дефиниран соченият тип.

Модификатор const

Модификаторът **const**, както е дефиниран в ANSI стандарта, забранява всяко присвояване на стойност на обекта и го предпазва от странични ефекти от операции върху него. Указателят **const** не може да бъде променян, въпреки че соченият от него обект може да се променя. Забележете, че модификатор **const**, използуван сам, е еквивалентен на **const int**.

Да разгледаме примера:

```
const  float pi     = 3.1415926;
const   maxint      = 32767;
const  *char str    = "Поздрави";
```

При тези декларации следните оператори са непозволени:

```
pi = 3.0;           /* Присвояване стойност на константа */
i  = maxint--;      /* Опит за промяна на константата    */
str = "Нов текст";  /* Опит за пренасочване на указателя */
```

Забележете, обаче, че обръщението **strcpy(str,"Нов текст")** е правилно, тъй като то копира символния низ в мястото, сочено от **str**.

Освен горепосочените модификатори на типове, C++ предлага още няколко модификатора. Това са: **extern**, **static**, **register**, **auto**, **pascal**, **cdecl**, **volatile**.

Модификатор auto

Модификаторът **auto** се използва за дефиниране на автоматични променливи. Типът **auto** е подразбиращ се.

Модификатор extern

Модификаторът **extern** се използва за дефиниране на външни функции и променливи. Използва се при разделното компилиране.

Пример:

```
#include <stdio.h>
extern int m;
void MyFunc()
{
  if( m != 0 ) printf("\n m is not null");
  else printf("\n m is null");
}
```

Модификатор static

Модификаторът **static** се използва за промяна времето за съществуване или видимостта на променливите.

Ако една променлива е глобална и е декларирана като **static** за нея важат следните особености:

- всички глобални променливи се записват в heap на програмата, следователно съществуват през цялото време на изпълнение на програмата. Т.е. модификатора **static** не действа върху времето на живот на променливата.
- Модификатора **static** променя видимостта на глобалната променлива. Това означава, че променливата може да се използва само в текущият модул и неможе да се декларира в друг модул като външна променлива.

Локална променлива:

- всички локални променливи се разполагат в стека на програмата. Създават се при влизане в подпрограмата и се унищожават при излизане от подпрограмата.
- Ако една променлива е декларирана като **static**, тя се разполага в **heap** на програмата, следователно времето и за живот е времето за изпълнението на програмата. Но видимостта на променливата не се променя т.е. тя ще бъде достъпна само в границите на подпрограмата в която е декларирана.

Модификатор register

Модификаторът **register** се използва в случаите, когато се търси по-голямо бързодействие на програмата. Ако една

променлива е декларирана като **register** и повреме на компилирането компилаторът прецени, че някой от регистрите на процесора е свободен, то той ще бъде резервиран за променливата.

Модификаторите **auto**, **extern**, **register** и **static**, ще бъдат разгледани по-подробно в следващите глави.

Модификатор volatile

Модификаторът **volatile**, също дефиниран в стандарта ANSI, е почти противоположност на **const**. Той показва, че обектът може да бъде променян не само от програмиста, но и от процеси извън програмата, като прекъсванията, нишки или входно/изходни портове. Обявяването на обект като **volatile** "предупреждава" компилатора да не прави приемания, отнасящи се до стойността на обекта по време на оценяването на съдържащия го израз, тъй като стойността му (теоретично) би могла да се промени всеки момент. Освен това, компилаторът няма да направи такава променлива регистъра.

Например:

```
volatile int ticks;
interrupt timer() {
        ticks++;
}

wait (int interval)  {
    ticks = 0;
    while (ticks < interval);
}       /* Празен оператор */
```

Тези функции (при положение че **timer** е правилно свързана с прекъсването за хардуерния часовник) ще осъществят пауза, определена от стойността на аргумента **interval**. Трябва да се отбележи, че оптимизиращ компилатор може да отхвърли променливата **ticks** в цикъла **while**, тъй като стойността ѝ не се променя.