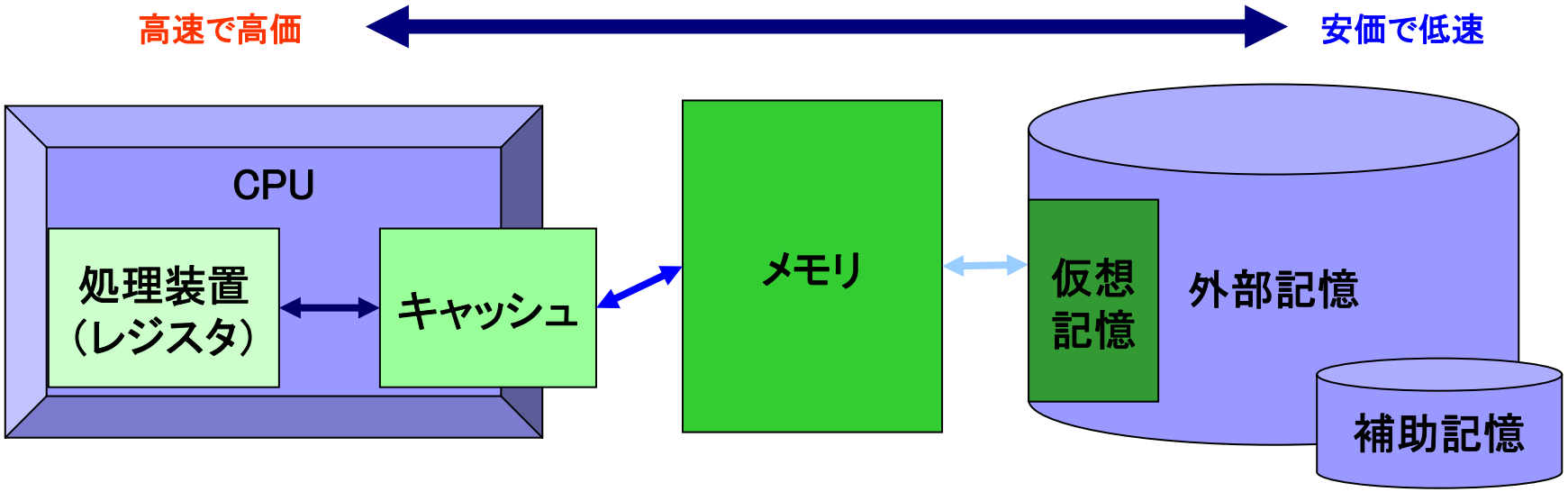
コンピュータ概論　第10回

記憶装置と配列

記憶装置（おさらい）

- 記憶
  - CPUで必要となるソフトウエア（プログラム・データ）を記録・保持すること
- 記憶装置
  - 主記憶装置：主にCPUが使用中の情報を保持
    - メモリ
    - キャッシュ
  - 外部記憶装置：主にファイルを保持
    - ハードディスク
    - フロッピーディスク

メモリの階層構造と効率（おさらい）

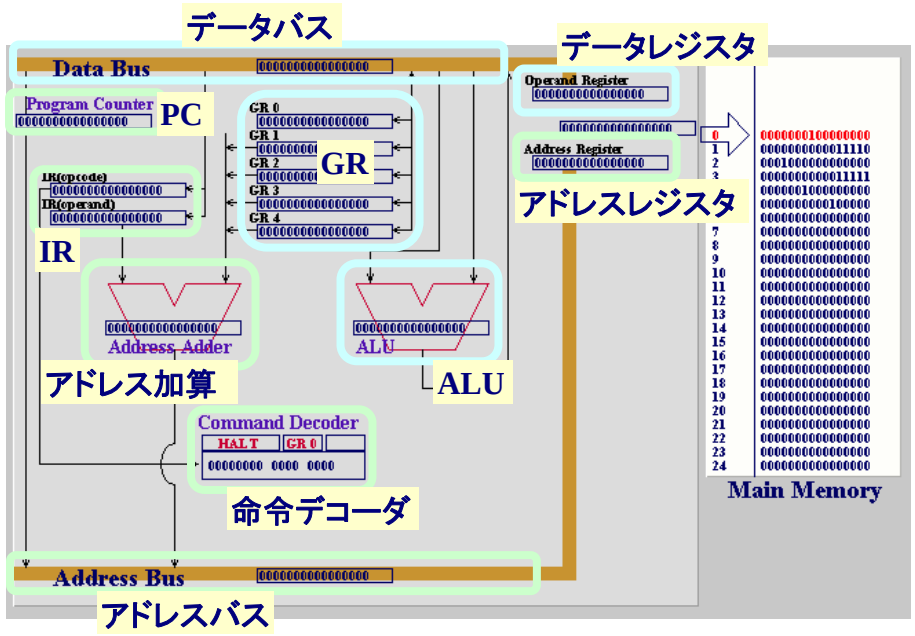


レジスタ＞キャッシュ＞メモリ＞仮想記憶（外部）

- 安さと動作効率のための工夫
- **キャッシュによる効率化**　　（欠点：高い）
  アクセス速度の速いキャッシュを用いて CPUは動作待ち時間を削減
- **仮想記憶による大容量化**　　（欠点：遅い）
  メインメモリを充分に用意できない場合に外部記憶装置で代用

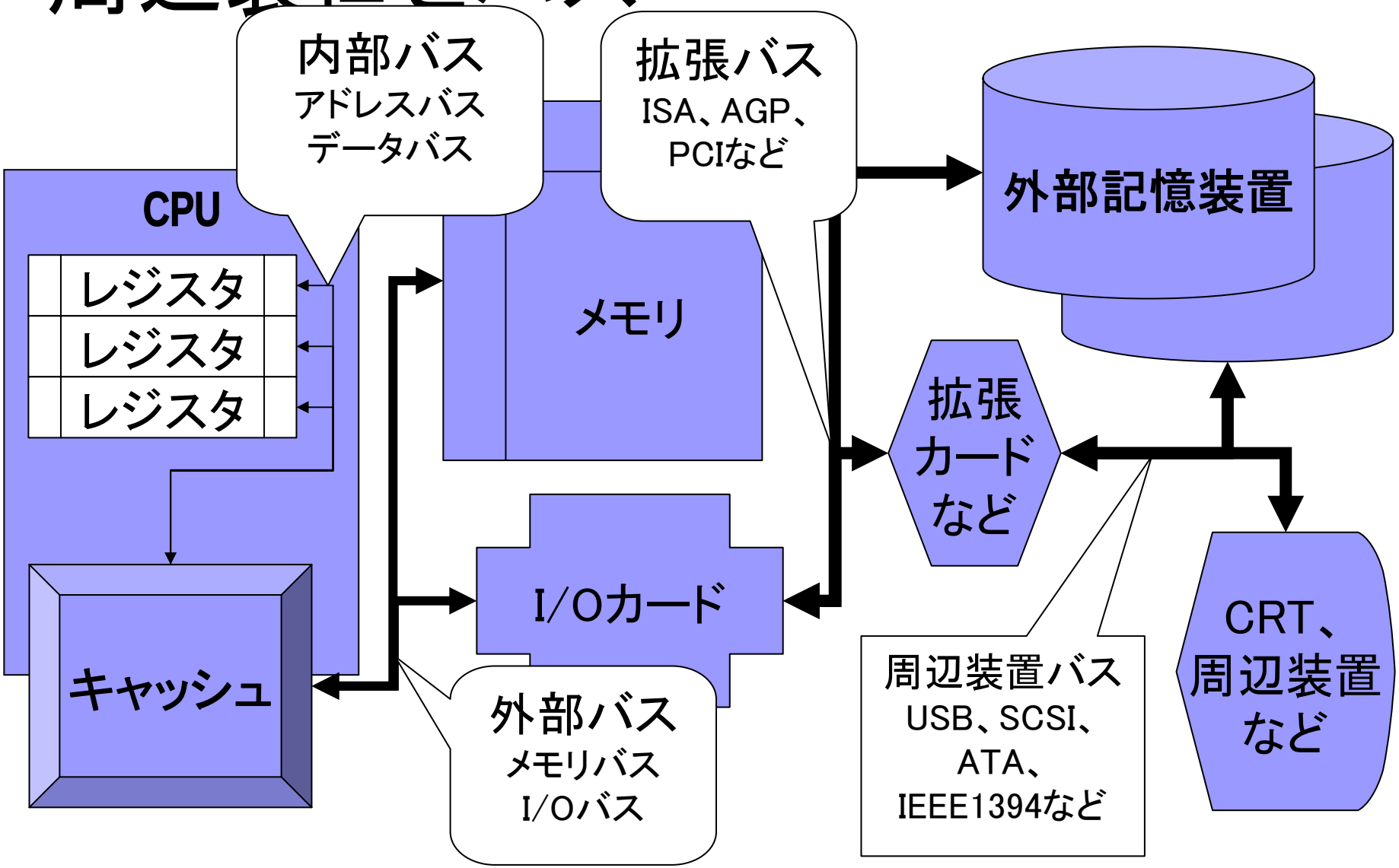
CPUと記憶装置の関係

CPUそのものは、プログラムやデータをほとんど保持できない
- 数ワード分のレジスタのみ



- プログラムやプログラムの動作で必要となるデータ
  - 必要に応じて記憶装置からCPU(レジスタ)へ送られる.
- CPUの動作した結果生じたデータやファイル
  - CPU(レジスタ)から記憶装置に送られ保持される.

周辺装置とバス（装置間のデータ通信路）



記憶装置と記憶サイズ

- レジスタ
  - CPU内の記憶装置（論理回路）
  - 速度：1命令サイクルで読み書きできる（数GHz）
  - 容量：１ワード（CPU命令のデータ量の単位）程度が数個
    - ワードという単位にはいろいろな意味があるので注意

- キャッシュ
  - CPUとメインメモリの中間
    - 内部キャッシュ（CPU内部に作りこまれたもの）
    - 外部キャッシュ（CPU外部の高速なSRAM）
    - １次、２次、３次と多階層の場合も（通常１次が内部キャッシュ）
  - 速度：１～数命令サイクル（１～数GHz程度）
    - 内部バス、メモリバスの速度にも依存
  - 容量：KBオーダー（32KB～1024KB）
  - 頻繁に使う情報をメインメモリからまとめて取り出しておく

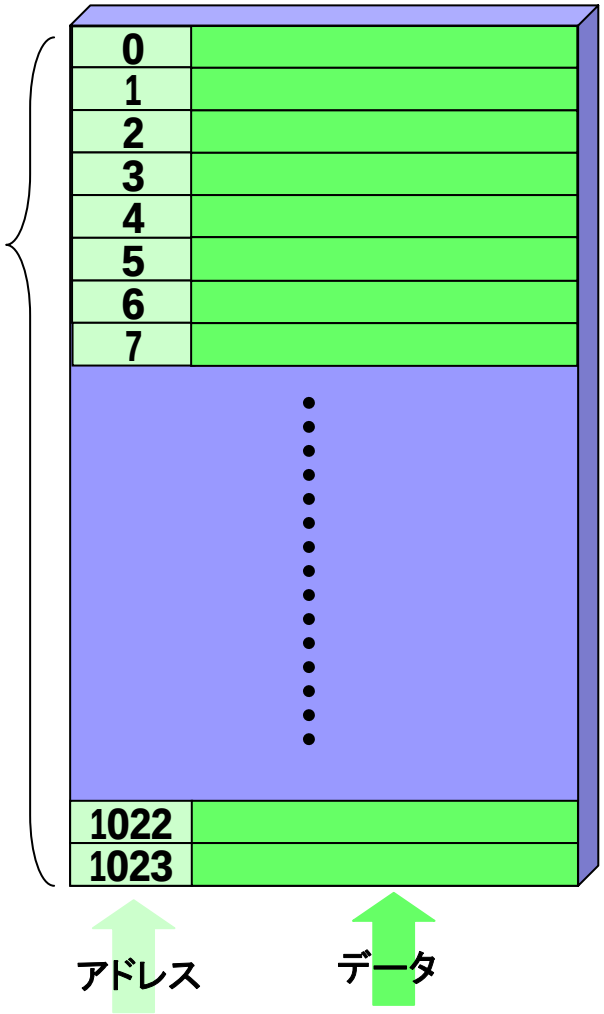
記憶装置と記憶サイズ

- メインメモリ
  - 主記憶装置の代表
  - 安価な記憶用LSI（DRAM）
  - 速度：数～数十命令サイクル（500MHz程度）
    - メモリバスの速度に依存
  - 容量：MBオーダー（128MB～2G(2048M)B程度）
  - CPUで実行するプログラムやデータを外部記憶から取り出しておく
- 外部記憶装置（補助記憶装置）
  - ハードディスクなど
  - 磁気媒体で非常に安価（非常に低速）
  - 速度：数十～数百命令サイクル（数十～100MHz程度）
    - 拡張バスの速度に依存
  - 容量：GBオーダー以上（40GB～数TB）
  - メインメモリに入りきらない場合のための仮想記憶としても使う

メモリ空間

- メモリ空間（アドレス空間）
  - CPUが理論的に扱い可能な主記憶領域
    - ワード×全アドレス分の記憶領域
    - CPUが処理中のソフトウェアを記憶
    - ファイル（外部記憶装置）は含まない

    メモリ空間のサイズが主記憶装置のサイズの上限値
- アドレス（番地）
  - メモリ空間上のひとまとまりのデータ（通常ワード単位）を指し示す、0から始まる整数の通し番号
  - メモリのデータ読み書きはアドレス単位で行なう
- ✓ワード（場合により意味が異なるので注意）
  - アドレスで特定されるひとまとまりのデータ
    - ビットやバイト単位の場合もある
  - CPUの命令やデータを構成するビット数
    - CPUによって異なる
    - 一般に2ワードで1命令、1ワードで1整数を構成
  - 2バイト

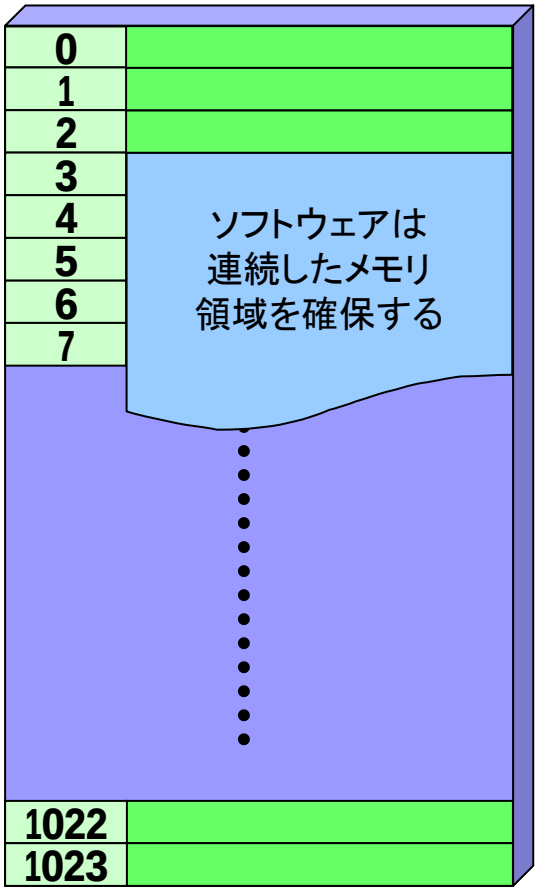


10ビット長のメモリ空間を持つ場合、理論上は0～1023番地までのアドレスが可能で、合計1024ワードのデータが記憶できることになる。

メモリの利用

- **メモリ空間**
  - 主記憶～仮想記憶に記憶
  - CPUの使用率が高い領域（ソフトウェア）はキャッシュに移動
- **ソフトウェアの配置**

  ひとまとまりのデータは連続する領域に配置
  （ディスク上のファイルのようにバラバラにならない）
  - プログラム
    - 実行中のものはメモリ上に配置
    - プログラム中の変数用の領域も含む
  - データ
    - プログラム中に含まれないデータ（特に画像データや音楽データなどの可変長データ）は必要に応じてプログラムが領域を確保
      （但しそのようにプログラムを作る必要がある）

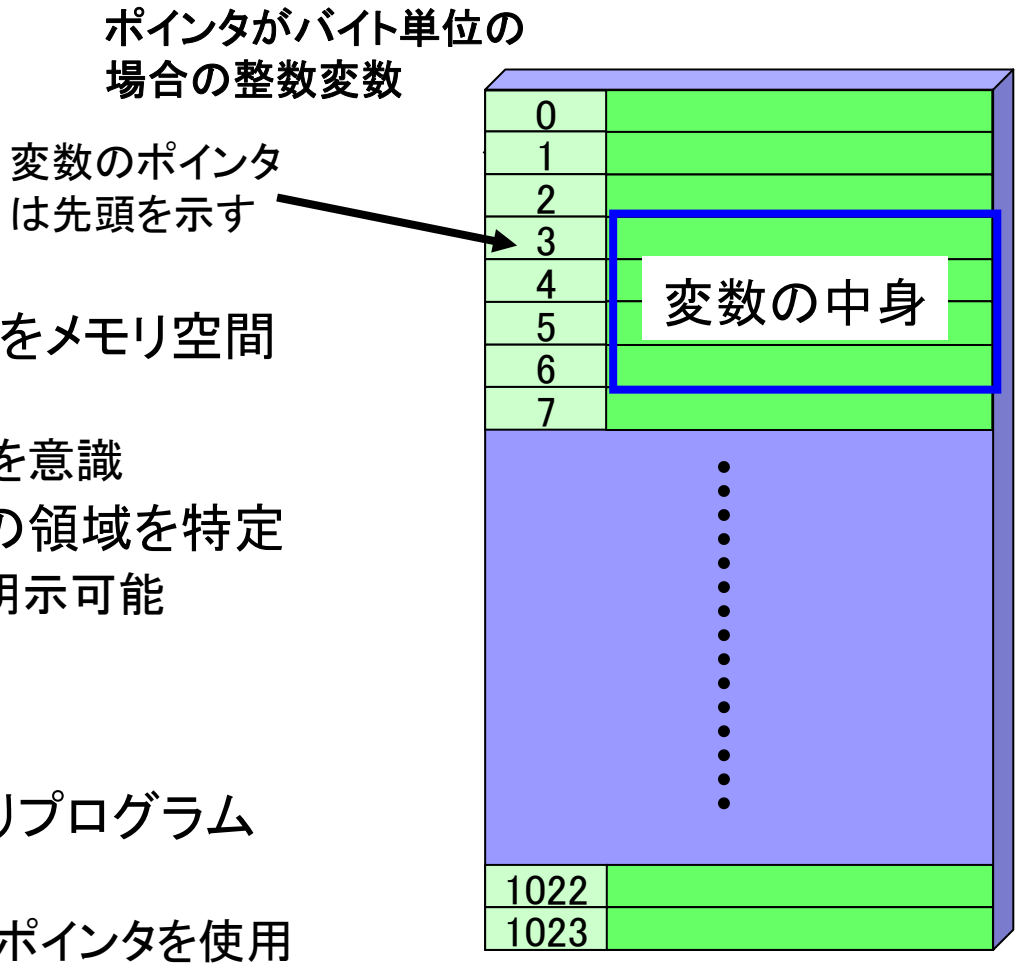


変数

- 高級言語的視点から見た変数
  - プログラムにおけるデータの入れもの
  - 型により変数（データ）の種類を区別
- 低級言語的視点から見た変数
  - データを記憶するためのメモリ空間上のある領域
  - 型により領域のサイズを決定
- 変数名はアドレスの代わり
  - 実際の記憶場所はアドレスで指定
  - 変数名を使うことで低級言語的なアドレスを意識しないで済む

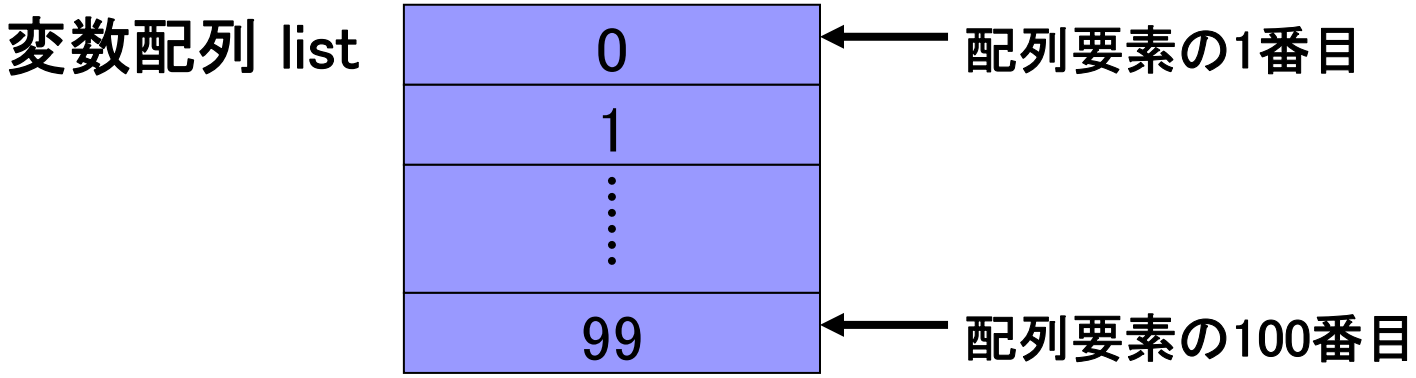
Cにおける変数

- Cでは低級言語的に変数を扱うことが出来る
- 変数
  - 型に合わせた領域（サイズ）をメモリ空間上に確保
    - sizeof演算子のように領域を意識
  - 変数名でメモリ空間上のその領域を特定
    - &演算子でそのポインタを明示可能
- ポインタ
  - Cにおけるアドレス情報
  - 変数だけでなく、関数（つまりプログラムの一部）も示すことが可能
    - scanfでは変数への代入にポインタを使用

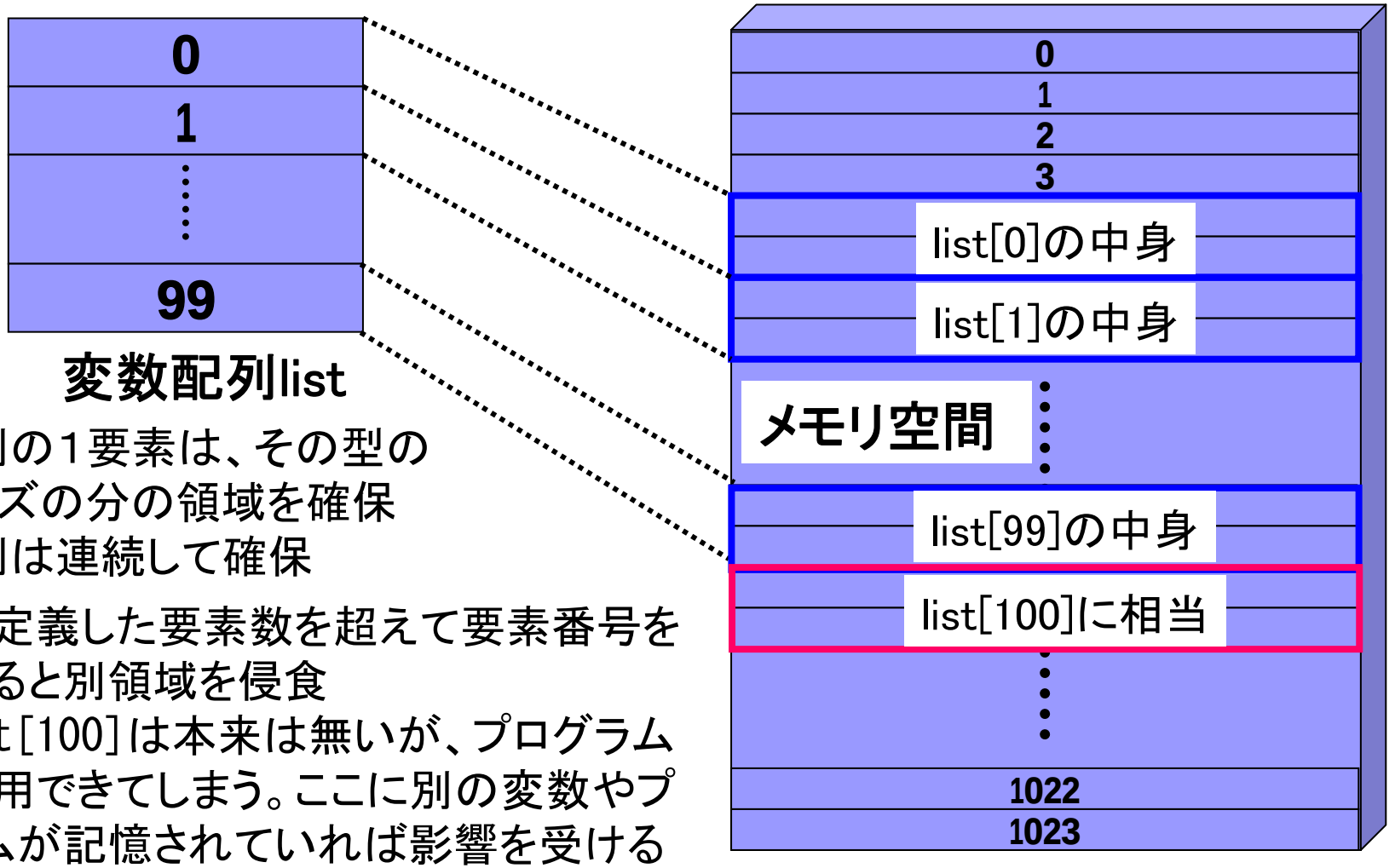


配列（変数配列）

- 変数を複数まとめて扱う手段
  - 1つの代表名を持つ
    - Cの場合、代表名は先頭要素のポインタを示す
  - それぞれの変数を配列要素と呼ぶ
    - 各配列要素のサイズは変数の型で決定
  - 要素番号でそれぞれの変数を指定する
    - Cの場合、要素番号は0からはじまる



メモリ空間における配列



- 配列の１要素は、その型のサイズの分の領域を確保
- 配列は連続して確保

（注意）定義した要素数を超えて要素番号を指定すると別領域を侵食
例）list[100]は本来は無いが、プログラム上は使用できてしまう。ここに別の変数やプログラムが記憶されていれば影響を受ける

配列の使い方

- 複数の情報をまとめて扱う
  - 数列の記憶
  - リスト（表など）の記憶
- 繰り返し処理との併用
  - まとめて同じような処理を行う
  - 順番に処理をする

配列の定義

- 変数配列の定義

変数型　代表名[要素数];

例）int ary[10];

- ary[0]～ary[9] の10個の整数変数の配列

- 要素数はプログラミング時に決定
  - 十分な数を確保する
    - プログラム実行中に不足しても簡単には増やせない
  - 必要以上に確保するとメモリが無駄になる
    - 確保したメモリ領域は他に流用できない
- 代入されていない要素の中身も空ではない
  - 多分何かのデータが入っている
  - 何が入っているかは実行するまで判らない（通常の変数も同じ）

例題1

- 配列を使ったプログラム（数列 $a_i=2i+3$）

```
#include<stdio.h>
main(void){
    int i, a[10];
    for(i=0; i<10; i++){
        a[i] = 2*i+3;
    }
    for(i=9; i>=0; i--){
        printf("%d番目の要素は%dです。¥n",i,a[i]);
    }
}
```

要素数は10（0～9）

iは0～9

例題2

- 配列を使ったプログラム(整数の総和)

```
#include<stdio.h>
main(void){
   int i, s=0, list[10];
   printf("10個の整数を入力してください¥n");
   for(i=0; i<10; i++){
       scanf("%d", &(list[i]));
   }
   for(i=0; i<10; i++){
       s += list[i];
   }
   printf("整数の合計は%dです。¥n",s);
}
```

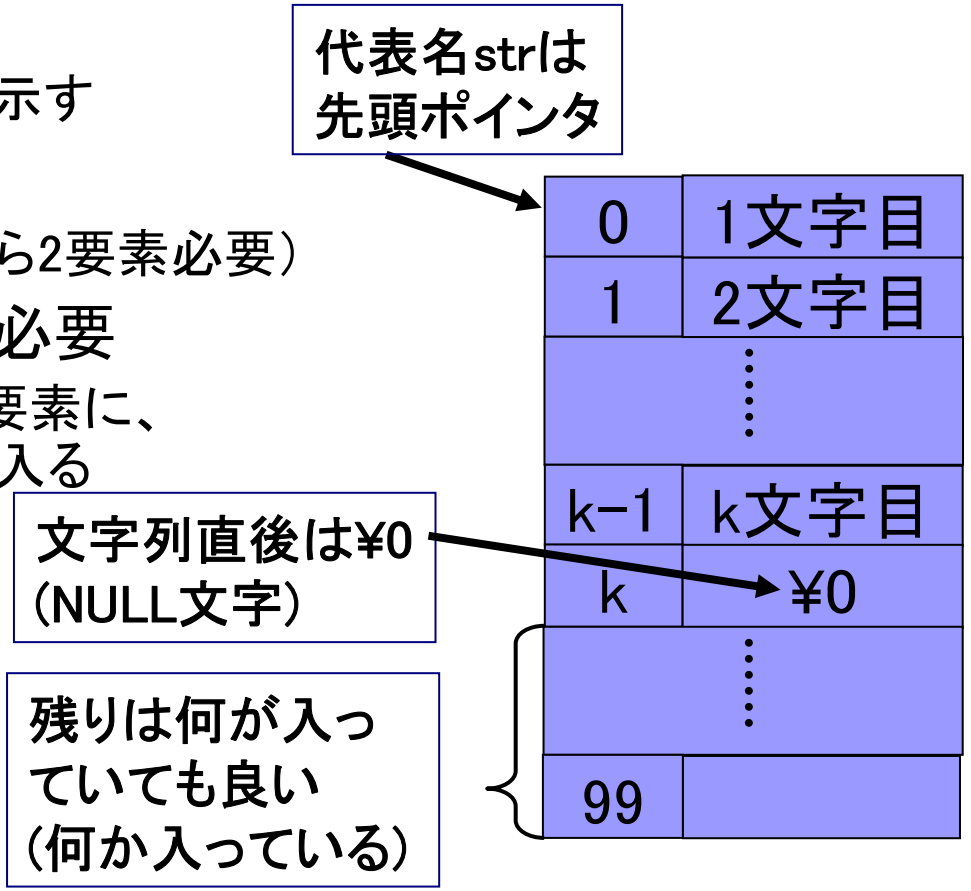
1つ1つの要素のポインタ(アドレス)を明示するために()を付けてから&

各要素の合計

文字列

- Cにおける文字列
  - 文字型変数の配列
    - 代表名は先頭のポインタを示す
  - 配列要素は文字型
    - 半角1文字（2バイトコードなら2要素必要）
  - 要素数は最大文字数＋１必要
    - 最後の文字（要素）直後の要素に、文字コード0（NULL文字）が入る

例）文字列 str[100]
k文字の文字列の場合



例題3

- 文字列を使ったプログラム

```
#include<stdio.h>
main(void){
  char name[10];
  printf("Input your name:");
  scanf("%s",name);
  printf("Hello, %s¥n",name);
}
```

文字列は最長9文字

代表名はポインタを示すので&は不要

文字列をまとめて表示する

例題4

- 文字列を文字型変数の配列として扱う（例題3を変更）

```
#include<stdio.h>
main(void){
  char name[10];
  int i;
  printf("Input your name:");
  for(i=0; i<10; i++){
     scanf("%c",&(name[i]));
  }
  for(i=0; i<10; i++){
     printf("%c",name[i]);
  }
}
```

1文字ずつ順番に
入力する(要改行)

1文字ずつ順番に
表示する

おさらい

- 記憶装置
  - CPUとの関係
  - キャッシュから仮想記憶まで
- メモリ空間
  - アドレスと変数の関係
- 変数配列
  - 複数の変数をまとめて扱う
  - 文字型配列による文字列

演習

- 文字列（10文字まで）を入力して、逆順に表示するプログラムを作成せよ。
  - 10文字に満たない場合は、何文字目に¥0があるかを調べて、そこから表示する。
    （strlen関数は使わないこと）
  - 入力（scanf）のときは%sを使うこと
- このプログラムを提出せよ

- 宿題：10文字以上入力するとどうなるか？